LIVERPOOL

SOFTWARE GAZETTE

MAY 1980

FOURTH EDITION 50p

During the Autumn of 1979 a small group consisting of successful publishers and acclaimed computer experts sat down to plan a new publication. Their brief? To produce the best all-round computer magazine on the market. Three months later the publication was born. It's name?

Computer Age

Computer Age is still only four issues old but already it is acknowledged by thousands to have achieved its original aim. Each new issue is enjoying increased sales and advertising revenue – but that's only part of the success story. More and more respected professionals are adding their names to the list of contributors and, one supposes, to an ongoing success story.

Whatever your interest in computers, the Publishers of Computer Age are confident you will be satisfied with the publication.

Why not put them to the test and see a sample copy? Just send 60p (to include postage and packing), the issue you would like to see (listed below) and full details to: COMPUTER AGE, 4 VALENTINE PLACE, LONDON S.E.1. A small price to pay for keeping in touch with the computer age!

ISSUE 1
Clive Jenkins and the unions' views. Are games playing programs intelligent? Artificial intelligence and the layman. Micro computers in education. Algorithmic designs. How mighty micros can aid super sales. Why not a computer in your business? Some microcomputer uses in psychology. Learning FORTH. Office of the future, and much more.

ISSUE 2
The government's views on computers. Cultural energy and the personal computer. Tribute to Christopher Evans. Microcomputers in the legal professions. Accountants' guide to microcomputers. The peripheral connection. Topics in artificial intelligence. Why not a computer in your school? Brand news, and much more.

ISSUE 3
Faceless bureaucrats – an invasion of privacy? Buying a personal computer. Statistical analysis on a micro. Doing it in binary. The small systems market hots up. Book reviews. When we're all ten years older. Database. The modular approach. Optimising employment.

ISSUE 4
The microcomputer and the teacher. Computer retailing. Using a digital ring. Desert island floppy discs. Computers for the smaller firm. Systems for small businesses. Keeping your computer busy. New technologies for education. More on FORTH.

"This periodical can be highly recommended on the evidence of its first two issues. It contains for the businessman, professional firm, teacher and individual a reliable guide almost equivalent to a computer Which on the ins-and-outs of buying and using computers."

The Times Monday February 18

CONTENTS

Page	Title
4	Editor & Publisher's Letter
6	Star Gate
14	Sharp Point
16	Letters to the Editor
20	Pets Corner
26	Apple Pips
30	Nascom Notes
32	Jet Set
41	Cartoon
42	6800 Processor Profile
44	A Forth Introduction
48	Pascal An Introduction
52	A Beginners Introduction to Pascal
55	A Useful Pascal Program
60	A Marvel Called the MC6809
66	A Number Processor on the Acorn
72	Architectural Software on the Cheap
75	Cartoon
76	Commercial Micro Software Fundamentals
80	Graphics Shape Translation
84	Social Effects of Micro Computers
86	Future Soft
88	Programming Practices
91	Etcetera

Editor

0.0

After the latest round of price cutting the cost of a home computer compares favourably with the cost of a domestic HI-FI and as all our readers are aware provides far more enjoyment. As we move into the microelectronics age we all have a personal duty to spread the gospel as widely as possible to create the broad base of understanding necessary for our future. In the coming decades computeracy will be added to numeracy and literacy as a fundamental requirement for everyday life and the sooner people face up to this and educate themselves the better.

Convince a friend to buy a computer.

0.1

The advent of the motor car at the beginning of this century caused a great deal of concern among social commentators of the day, they predicted that the change from labour intensive horse transport to the mechanised car would lead to widespread unemployment. They could see that grooms, blacksmiths etc., would no longer be needed and no way the motor car could provide alternative employment.

The motor car is now fully integrated into the infrastructure of our society and in western countries is the largest single factor in providing employment.

The average person has acquired a degree of personal mobility unheard of at any other time in history and by any measure the overall quality of life has increased enormously.

0.2

This issue of the magazine shows the first signs of reader participation with a letters page as well as an article and parts of both nascom notes and apple pips submitted to us by readers. We hope this involvement will continue and possibly increase even if we have to fuel it by being contentious or including deliberate mistakes.

0.3

There are now five high level languages freely available for most microcomputers, these are BASIC, Forth, Lisp, Pascal and Pilot and all five are, or will be, supported by this journal. There is currently a great deal of debate concerning the relative merits and demerits of these languages and if LSG can help fuel these arguments, so much the better.

There is still a need for a general, easy to use, structured language that can be used as an interpreter during program development and as a compiler when the program is complete. There is a further need for a simple I/O oriented language for use in the industrial control environment. When and how will these needs be filled? What happened to APL on Microcomputers?

0.4

Up and coming articles cover the following topics, among others;-
Crystals Z80 BASIC, LISP, Sorcerer WP Pac, follow up to JET SET, and a CESIL interpreter. The number of regular features is also increasing to cover more machines and more topics.

B. Everiss

BRUCE EVERISS

Publisher

1.0

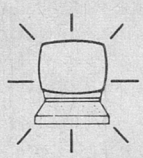
With this issue we reach 100 pages, a level which we will not be able to exceed until we have far more advertising revenue. Meanwhile the number and quality of articles continue to be in excess of those of any other microcomputer journal published in the U.K.

Computer Bookshop Gives MICROBOOK Retailers

FAST, FREE DELIVERY Ex Stock!

And the best selection of MICRO-COMPUTER books in the business

Send for full details of all our BIG Selling Books NOW!!!



The Computer Bookshop

Temple House, 43/48 New Street,
Birmingham B2 4LA.
Telephone: 021-643 4577.

Buy with confidence from the specialists

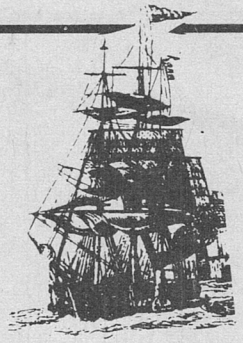
HB COMPUTERS LTD

FOR THE BEST IN

SHARP MZ80K SOFTWARE

Send For Our Free Catalogue

HB COMPUTERS LTD
22 NEWLAND STREET, KETTERING
NORTHANTS.
Tel. (0536) 83922 & 520910 Telex 341297



LIVERPOOL SOFTWARE GAZETTE

Editor/Publisher: Bruce Everiss.
Editorial Assistant: Nikki Devereux.
Contributing Editors: John Stout, Dr Martin Beer, Dave Straker.
Advertising/Subscriptions: Nikki Devereux.
Artwork: Peter Cross, John Burrows.

ADVERTISING:

Full page (17.5 cm x 24 cm)	£180.00
Half page (Upright 8.5 cm x 24 cm)	£ 95.00
Half page (Landscape 17.5 cm x 11.75 cm)	£ 95.00
Quarter page (8.5 cm x 11.75 cm)	£ 52.00

Agency discount 10%

Please contact Nikki Devereux on 051-227 2535 if you would like further details.

DISCLAIMER: 'All the information in the magazine has been throughly debugged and tested. However, no guarantees are made as to its truth or validity'.

TRADE DISTRIBUTION:— Computer Bookshop, 43-45 Temple Street, Birmingham, 021-643 4577.

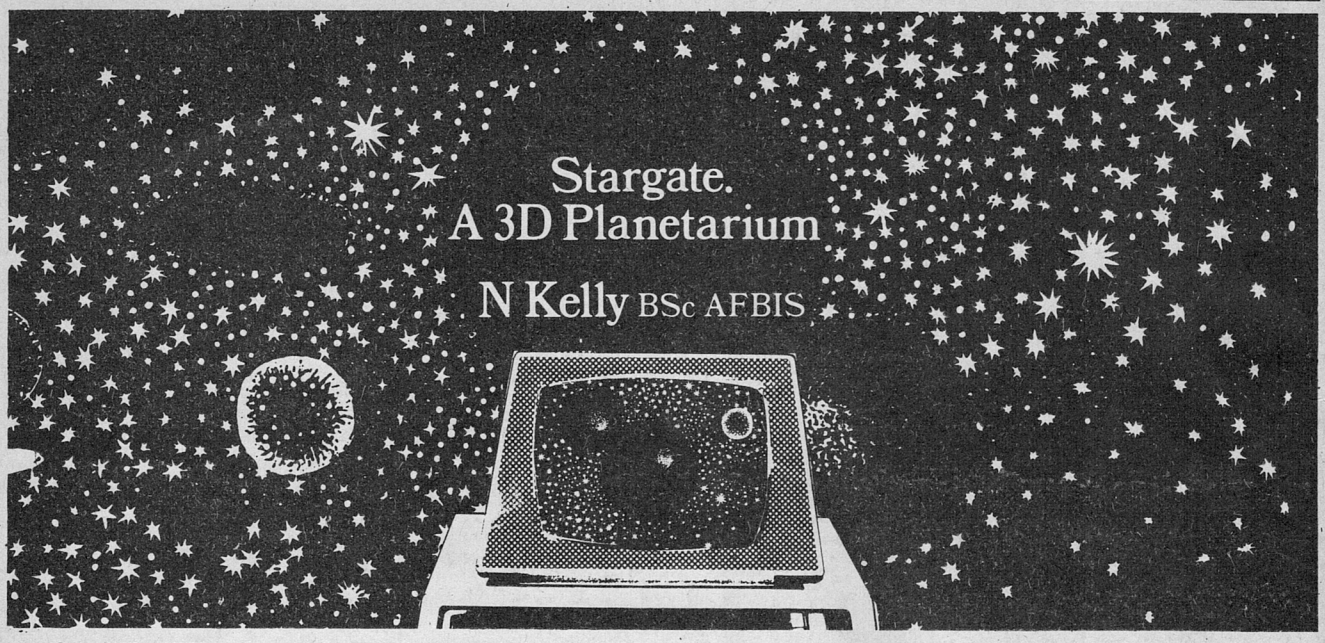
U.S. DISTRIBUTION:— Bits Inc., P.O. Box 428, 25 Route 101 West, Peterborough, NH 03458.

(c) LIVERPOOL SOFTWARE GAZETTE 1980

THE LIVERPOOL SOFTWARE GAZETTE is published bi monthly by Liverpool Software Gazette Limited, 14 Castle Street, Liverpool L2 0TA. Registered in England No 1477285

Subscriptions: Within Great Britain, £6.00 for 12 issues. Individual copies, by post 60p. Please tell us the issue you would like to start a subscription with!

REPRINTS: Articles that are explicitly marked as having restricted reproduction rights may not be copied or reprinted without written permission from Liverpool Software Gazette. All other articles may be reprinted for any non-commercial purpose provided a credit line is included stating that said material was reprinted from the Liverpool Software Gazette, 14 Castle Street, Liverpool. L2 0TA. Please send copies of any reprints to Liverpool Software Gazette, attention of Bruce Everiss.



YOU CAN SET;
MODES ; MAP, VIEW, TANK
LOOK ; F,B,U,D,L,R
SCALE ; 1 to 10 & 'A' to 32
X,Y,Z OFFSETS BETWEEN + &- 100 ly
'DEPTH OF VIEW', TO 999 ly
'PAN' AND 'TILT' WITH 'GYRO'
AUTO-BRIGHTNESS CONTROL, FOR DIM STARS.
ALSO, 'JUMP' LOCATES SELECTED STARS,
PREDICTS AND PLOTS INTERSTELLAR ROUTES.

YOU MAY SET CONTROLS WITH INITIALS.
Use:-
'C' FOR ABC 'H' FOR HELP
'J' FOR JUMP 'S' FOR STOP
'G' FOR GYRO 'O' FOR ARRAY READER
'N' FOR NEAR ('DEPTH OF VIEW')
'A' FOR 'ADD 10 TO SCALE'
' ' TO OFFSET TANK CENTRE
'K' FOR CATALOG
AND 'Q' CUES YOUR CHANGES.

TO CONTINUE, TYPE 'Y' FOR YES .....

THE PROGRAM

STARGATE 100 manipulates the 3D co-ordinates of 101 'nearby' stars, and presents them on the Apple Hi-res screen in several forms, including a pseudo-3D TANK. A Status Board occupies the four-line text window, displaying Stargate variables and active options.

The program consists of a "preset" section which defines and fills strings and variables, two pages of instructions, the status board and the quick-set 'menu selector' with its associated sub-routines. Many of these are 'number input' extensions of the menu, but the others fall into three groups.

'Q' starts the 'Display' loop, which calls 'Look/Mode' and 'Mag-plot' to sort and display acceptable stars. 'Array reader' and 'Catalog' supply data to screen or printer. 'JUMP' complements 'Display', using the 'Look/Mode' sub-routine to blink selected stars, calculate the true distance between them, and over-write the existing star-field with your route.

STARGATE 100 is a slimmed version of my '600' program, which holds the majority of stars above 4th visual magnitude, shows the constellations, and is proportionately slower. Together with its extended array, it is available on disc from Microdigital.

THE STORY

In 1978, I wanted a map of the nearest stars. I could n't find one. A search in Liverpool's renowned Central Library supplied data from which my 'Sinclair' calculated XYZ co-ordinates. Alpha, Prox Centauri, Barnard's star (with planets!), Sirus, Epsilon Eridani, Tau Ceti, Procyon, Epsilon Indi ...Lovely names. I plotted them on an A4 sheet.

'Allen's Astrophysical Quantities' gave all 72 systems within 20 ly, so I transferred my stars to a wall-hung A3 graph. I added an acetate overlay for depth, then kept searching. Enquiries led me to the 'Atlas Coeli' catalog of visual stars, and my map began to fill. Midway, and shopping-hours before the Christmas break, my calculator expired.

Infected by my brother's copy of 'Illustrating BASIC' I bought a TI 57 Programmable. I also bought a 'Hobby Electronics' which featured 'simple programming' and the British Inter-planetary Society's 'Starship Daedalus' project. (BIS ? Oh, yes; They designed a manned Lunar lander in 1939 and got it right, but more on them later ...)

Thus armed, I reworked my lists and caught a few rogues. I also cross-checked my sources, and found significant differences. Seeking a single source, I bought a copy of the 'Daedalus project', and was so impressed that I joined the BIS. I asked their advice, and was recommended to a publication of the Royal Greenwich Observatory.

To my delight, it contained the 1744 nearest star systems for £5 + VAT. However, the mere thought gave me 'calculator's finger' and 'beady-LED eye', set

me itching for a VDU and proper keys. Systems with adequate graphics and memory started at £2k, but prices were falling. I brushed up my BASIC and, come summer, I took a note-book on holiday. In a beach-fringing Greek olive grove, STARGATE was born.

In September, I bought a B&W APPLE II+, 32k, APPLESOFT-in-ROM and the learning process began.

My variables spill into HGR! LOMEM : 16384. My variables get chewed! Add 16k. My stars are upside-down and back-to-front! Oops, my fault. I'd transposed Sin/Cos in a formula -- the year before! (My lists! My graphs!) My program looked forwards to the **right!** Again, my fault; Sloppy thinking in 'Look/Mode'.

By now, it was December, and STARGATE was approaching its present form. I kept changing things, though. Adding options, trying everything else-how, was filling two or three tapes a week, excluding over-writes. I winced at the cost, got a disc-drive and kept it busy.

Eventually, 'Gyro' pushed the program into HGR. 'Out with the REMs!' I re-wrote the instructions, added 'Tank offset', and again lost the 'Array reader'. This was serious; What could I cut? What was left? Then I noticed that, within PRINT statements, the APPLE concatenates without semi-colons, and so freed enough bytes to finish.

Program execution time remains a sore point. I've done my best, but '600's 590 stars take 90 seconds, or 4 minutes with 'Gyro'. I'm learning Assembler, and dreaming of an Applesoft compiler or super-fast arithmetic board to hasten the first hundred lines for that ultimate, a real-time display.

Meanwhile, I've an option called 'Neighbours', which lists the nearest stars to a given star or point by range. It's Pythagoras plus 'Super Sort' (LSG Jan'80), but it just won't fit.....

But that's STARGATE.

N. KELLY. B.Sc AFBIS

The British Interplanetary Society is at:
27/29 South Lambeth Road,
London SW8 1SZ

★ Their monthly 'Journal' and 'Spaceflight' may be found in many libraries and they welcome enquiries.

```
1  GOTO 200: REM STARGATE
2  REM BY N.KELLY 7/3/80
3  REM C/O MICRODIGITAL
4  REM STARS C/O ALLENS ASTROPHYS'ED 3
6  REM BIS @ 27 S'TH LAMBETH RD
7  REM LONDON SW8 1SZ
9  REM LOOK/MODE
10 OO = 0:X = X - PX:Y = Y - PY:Z = Z - PZ
20  ON L + 1 GOTO 21,22,23,24,25,26
21 XI = X:X =  - Y:Y = Z:Z =  - XI: GOTO 27
22 XI = X:X = Y:Y = Z:Z = XI: GOTO 27
23 XI = X:X =  - Y:Y =  - XI:Z =  - Z: GOTO 27
24 XI = X:X =  - Y:Y = XI: GOTO 27
25 YI = Y:Y = Z:Z =  - YI: GOTO 27
26 X =  - X:YI = Y:Y = Z:Z = YI
27  IF TI AND (Z OR Y) THEN R =  SQR (Z * Z + Y * Y):SO = Y / R:CO =
     Z / R:Y = R * (SO * CT + CO * ST):Z = R * (CO * CT - SO * ST)
29  IF (PA) AND (X OR Z) THEN R =  SQR (X * X + Z * Z):SO = Z / R:CO =
     X / R:X = R * (CO * CP - SO * SP):Z = R * (SO * CP + CO * SP)
30  IF (Z >  = 0) OR (NE AND Z < NE) THEN OO = 1: RETURN
31  ON M + 1 GOTO 34,32,33
32 X = 10 * X /  ABS (Z):Y = 10 * Y /  ABS (Z): GOTO 34
33 X = X * (1 + Z / 100):Y = Y * (1 + Z / 100)
34 X =  INT (S * X) + 140
36  IF X < 1 OR X > 278 THEN OO = 1: RETURN
38 Y = 80 -  INT (S * Y)
40  IF Y < 2 OR Y > 157 THEN OO = 1
42  RETURN
49  REM MAGPLOT
50 OO = 0:F =  INT (F + J + K *  LOG ( - Z)): IF F <  - 1 THEN F =  -
    1
56  IF F > 5 AND C = 1 THEN F = 5
58  IF F > 5 THEN OO = 1: RETURN
60  ON 6 - F GOTO 74,72,70,68,66,64,62
```

```
62  HPLOT X - 1,Y - 1 TO X,Y - 2: HPLOT X,Y - 2 TO X + 1,Y - 1
64  HPLOT X,Y + 2: HPLOT X,Y - 1
66  HPLOT X - 1,Y TO X - 1,Y + 1
68  HPLOT X,Y + 1
70  HPLOT X + 1,Y + 1
72  HPLOT X + 1,Y
74  HPLOT X,Y: RETURN
100  REM DISPLAY
105  IF M < 2 THEN 130
110 X = TX:Y = TY:Z = TZ: GOSUB 10
112  IF OO = 1 THEN  PRINT : PRINT "TANK WILL NOT WORK AS ": PRINT TX
     ","TY","TZ" IS OFF SCREEN.": FOR T = 0 TO 2500: NEXT : GOTO 300
115  HGR : HCOLOR= 7:XA = X:YA = Y: FOR N = 0 TO 100
120 X = A%(1,N):Y = A%(2,N):Z = TZ
122  GOSUB 10: IF OO THEN 160
127 XB = X:YB = Y: GOTO 132
130  HGR : HCOLOR= 7: FOR N = 0 TO 100
132 X = A%(1,N):Y = A%(2,N):Z = A%(3,N):F = A%(0,N) / 10
135  GOSUB 10: IF OO THEN 160
140  GOSUB 50: IF OO THEN 160
150  IF M < 2 THEN 160
152  HPLOT X,Y TO XB,YB: HPLOT XB,YB TO XA,YA
160  PRINT N: NEXT : PRINT "ARRAY FINISHED": FOR T = 0 TO 1000: NEXT
     : GOTO 300
200  REM PRESETS
203  LOMEM: 16384
207 X = Y:Z = XI:YI = SP:CP = ST:CT = R:SO = CO:OO = NE
210 S = 1:TX = TY:TZ = 0
250  DIM L$(5)
255 L$(0) = "F":L$(1) = "B":L$(2) = "U":L$(3) = "D":L$(4) = "L":L$(5)
     = "R"
260  DIM M$(2):M$(0) = "MAP":M$(1) = "VIEW":M$(2) = "TANK"
265 Q$ = "TO CHANGE, TYPE Y FOR YES, N FOR NO"
270 J =  - 7.6:K = 2.2
280  DIM C$(1):C$(1) = " WITH ABC"
281  ONERR  GOTO 298
282  DIM A%(3,100)
284 DI$ =  CHR$ (4):CI$ = "STARS 100"
285  PRINT DI$
287  PRINT DI$"OPEN"CI$
289  PRINT DI$"READ"CI$
291  FOR N = 0 TO 100
292  FOR T = 0 TO 3
293  INPUT A%(T,N)
294  NEXT T,N
295  PRINT DI$"CLOSE"CI$
297  GOTO 500
298  PRINT "OUCH !!  TYPE 'H' FOR HELP !!": PRINT  PEEK (222): FOR T =
     0 TO 2500: NEXT
300  IF A%(0,0) = 0 THEN  TEXT : HOME : PRINT : PRINT "ARRAY IS EMPTY
     ": PRINT : PRINT "'RUN' TO REFILL": END
301  ONERR  GOTO 298
302  PRINT "LOOK "L$(L)" AT ZOOM "S;C$(C)
305  IF PA OR TI THEN  PRINT " PAN IS "P". TILT IS "T1". ";
306  IF NE THEN  PRINT "'DOV' = " ABS (NE)" LY";
308  PRINT : PRINT "MODE IS "M$(M)" FROM "PX","PY","PZ
```

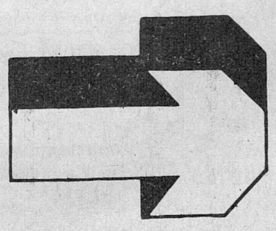
```
309  IF M = 2 THEN  PRINT "CENTRE "TX", "TY", "TZ" @ "NN" ";
310  GET V$: PRINT : FOR N = 0 TO 5: IF V$ = L$(N) THEN L = N: GOTO 3
     00
320  NEXT
330  IF V$ = "C" AND C = 0 THEN C = 1: GOTO 300
335  IF V$ = "C" AND C = 1 THEN C = 0: GOTO 300
360  IF V$ = "A" THEN S = S + 10 * (S <  = 22): GOTO 300
400  FOR N = 1 TO 3: IF V$ =  MID$ ("MVT",N,1) THEN M = N - 1: GOTO 3
     00
410  NEXT
440  IF V$ = "Q" THEN  GOTO 100
460  FOR N = 1 TO 10: IF V$ =  MID$ ("NJGHXYZOK£",N,1) THEN : ON N GOTO
     645,700,600,505,565,575,585,900,592,655
465  NEXT
492  IF  VAL (V$) >  = 0 AND  VAL (V$) <  = 9 THEN S =  INT ( VAL (V$
     )): IF S = 0 THEN S = 10
493  IF V$ <  > "S" THEN 300
494  PRINT : PRINT "TO RESTART,  'GOTO 500'.": FOR T = 0 TO 2500: NEXT
500  TEXT : HOME : PRINT : PRINT : HTAB 14: PRINT "STARGATE": PRINT :
      PRINT : PRINT : PRINT "          A 3-D PLANETARIUM."
503  PRINT : PRINT : PRINT : PRINT : PRINT : PRINT : PRINT "   TO CON
     TINUE, TYPE 'Y' FOR YES.": GET V$: IF V$ <  > "Y" THEN  PRINT :
     PRINT "TO RESTART,'GOTO 500'.": END
505  TEXT : HOME : PRINT : PRINT " YOU CAN SET :-": PRINT : PRINT "MO
     DES;   MAP, VIEW, TANK": PRINT : PRINT "LOOK ;   F, B, U, D, L,
     R"
510  PRINT : PRINT "SCALE;  1 TO 10 & 'A' TO 32": PRINT : PRINT "X,Y,
     Z OFFSETS BETWEEN +&- 100 LY"
512  PRINT : PRINT "'DEPTH OF VIEW', TO 999 LY": PRINT : PRINT "'PAN'
      & 'TILT', WITH 'GYRO'": PRINT : PRINT "AUTO-BRIGHTNESS CONTROL,
      FOR DIM STARS."
515  PRINT : PRINT : PRINT " ALSO, 'JUMP' LOCATES SELECTED STARS,":
     : PRINT "PREDICTS AND PLOTS INTERSTELLAR ROUTES."
517  PRINT : PRINT : PRINT " TO CONTINUE, TYPE 'Y' FOR YES.  ";: GET
     V$: IF V$ <  > "Y" THEN 500
520  HOME : PRINT : PRINT " YOU MAY SET THE CONTROLS WITH INITIALS.":
      PRINT "   USE :-"
522  PRINT : PRINT : PRINT " 'C' FOR ABC       'H' FOR HELP": PRINT :
     " 'J' FOR JUMP     'S' FOR STOP"
523  PRINT : PRINT " 'G' FOR GYRO      'O' FOR ARRAY READER": PRINT :
     " 'N' FOR NEAR  ('DEPTH OF VIEW')": PRINT : PRINT " 'A' FOR 'ADD
      10 TO SCALE'"
524  PRINT : PRINT " '£' TO OFFSET TANK CENTRE": PRINT : PRINT " 'K'
     FOR CATALOGUE": PRINT : PRINT "  AND 'Q' CUES YOUR CHANGES."
530  PRINT : PRINT : PRINT "TO CONTINUE, TYPE 'Y' FOR YES.  ";: GET V
     $: IF V$ <  > "Y" THEN 505
535  PRINT : GOTO 300
561  PRINT "TOO BIG !!"
563  PRINT "ENTER X": INPUT PX: IF  ABS (PX) > 100 THEN 561
565  PRINT : PRINT "X OFFSET IS "PX: PRINT Q$: GET V$: IF V$ = "Y" THEN
     563
571  PRINT "TOO BIG !!"
573  PRINT "ENTER Y": INPUT PY: IF  ABS (PY) > 100 THEN 571
575  PRINT : PRINT "Y OFFSET IS "PY: PRINT Q$: GET V$
577  IF V$ = "Y" THEN 573
580  GOTO 300
```

```
581  PRINT "TOO BIG !!"
583  PRINT "ENTER Z": INPUT PZ: IF  ABS (PZ) > 100 THEN 581
585  PRINT : PRINT,"Z OFFSET IS "PZ: PRINT Q$: GET V$
587  IF V$ = "Y" THEN 583
588  GOTO 300
590  PRINT "UNCATALOGUED !!"
591  FOR T = 0 TO 2500: NEXT
592  PRINT : PRINT "STAR POSITIONS": PRINT "ENTER CAT.NUMBER, OR 'E'
     TO ESCAPE.": INPUT V$: IF V$ = "E" THEN 300
593 T =  VAL (V$): IF T < 0 OR T > 100 THEN 590
594  PRINT : PRINT T"  "A%(0,T) / 10"  "A%(1,T)"  "A%(2,T)"  "A%(3,T)
     : GOTO 591
600  REM GYRO
605  PRINT : PRINT "PAN IS "P".  TILT IS "T1: PRINT Q$: GET V$: IF V$
      <  > "Y" THEN 300
610  GOTO 615
611  PRINT " TOO BIG !!"
613  PRINT "ENTER PAN, WITHIN +&- 45 DEG.": INPUT P:P =  INT (P): IF
      ABS (P) > 45 THEN 611
615  PRINT : PRINT "PAN IS "P: PRINT Q$: GET V$: IF V$ = "Y" THEN 613

618 SP =  SIN (P * .01745):CP =  COS (P * .01745):PA = 1: IF P = 0 THEN
     PA = 0
620  GOTO 625
621  PRINT " TOO BIG !!"
623  PRINT "ENTER TILT, WITHIN +&- 60 DEG.": INPUT T1:T1 =  INT (T1):
      IF  ABS (T1) > 60 THEN 621
625  PRINT : PRINT "TILT IS "T1: PRINT Q$: GET V$: IF V$ = "Y" THEN 6
     23
628 ST =  SIN (T1 * .01745):CT =  COS (T1 * .01745):TI = 1: IF T1 = 0
      THEN TI = 0
630  GOTO 605
640  REM DOV
642  PRINT : PRINT "SET 'DEPTH OF VIEW'. 0 DISCONNECTS.": INPUT V$:NE
      =  -  INT ( ABS ( VAL (V$)))
643  IF NE <  - 999 THEN  PRINT : PRINT "TOO BIG !!": GOTO 642
645  IF NE = 0 THEN  PRINT : PRINT "'DEPTH OF VIEW' IS OFF.": GOTO 64
     9
648  PRINT : PRINT "'DEPTH OF VIEW' IS " ABS (NE)" LY."
649  PRINT : PRINT Q$: GET V$: IF V$ = "Y" THEN 642
650  GOTO 300
654  REM £
655  PRINT : PRINT : PRINT "TANK CENTRE IS "TX", "TY", "TZ" @ "NN:
     : PRINT Q$: GET V$: IF V$ <  > "Y" THEN 300
658  PRINT : PRINT "SET TO STAR 'S', POSITION 'P' ": PRINT : PRINT "O
     R 'E' TO ESCAPE.": GET V$: IF V$ = "S" THEN 665
660  IF V$ = "P" THEN 672
661  GOTO 655
662  PRINT : PRINT "TOO BIG !"
665  PRINT : PRINT "ENTER SG NUMBER": INPUT N:N =  INT ( ABS (N)): IF
     N < 0 OR N > 100 THEN 662
667 TX = A%(1,N):TY = A%(2,N):TZ = A%(3,N):NN = N: GOTO 655
670  PRINT : PRINT "TOO BIG !"
672  PRINT : PRINT "ENTER XYZ CO-ORDS": INPUT TX,TY,TZ: IF  ABS (TX) >
     100 THEN 670
675  IF  ABS (TY) > 100 THEN 670
```

```
678  IF  ABS (TZ) > 100 THEN 670
680 NN = 0: GOTO 655
700  PRINT : PRINT ,"'JUMP'": PRINT : PRINT "PREDICTS AND PLOTS INTER
     STELLAR ROUTES.": FOR T = 0 TO 2500: NEXT
701 JN = 0
702  PRINT : PRINT "TO CONTINUE WITH JUMP, TYPE 'Y' FOR YES."
704  GET V$: IF V$ <  > "Y" THEN 300
708  IF JN = 0 THEN  PRINT : PRINT "ENTER FIRST STARGATE CATALOGUE NU
     MBER."
709  IF JN THEN  PRINT : PRINT "ENTER NEXT NUMBER."
710  INPUT N: IF N < 0 OR N > 100 THEN  PRINT "UNCATALOGUED !!": GOTO
     708
712 X = A%(1,N):Y = A%(2,N):Z = A%(3,N): GOSUB 10
714  IF OO = 1 THEN  PRINT "OFF SCREEN.  RE-ENTER NUMBER.": GOTO 708
740  IF JN THEN :J =  SQR ((A%(1,N) - A%(1,NA)) ^ 2 + (A%(2,N) - A%(2
     ,NA)) ^ 2 + (A%(3,N) - A%(3,NA)) ^ 2)
741 J =  INT (J + 0.5)
742  IF JN THEN  PRINT : PRINT "JUMP "NA" TO "N" IS "J"LY. OK? Y,N OR
      E."
743  IF JN = 0 THEN  PRINT : PRINT "FIRST IS "N" , OK? Y,N,OR E TO ES
     CAPE.":NA = N:XA = X:YA = Y
744  HCOLOR= 7: HPLOT X,Y: HPLOT XA,YA
746  IF  PEEK ( - 16384) > 127 THEN 750
748  FOR T = 0 TO 150: NEXT : HCOLOR= 0: HPLOT X,Y: HPLOT XA,YA: FOR
     T = 0 TO 150: NEXT : GOTO 744
750  GET V$: IF V$ = "Y" THEN  HPLOT XA,YA TO X,Y:NA = N:XA = X:YA =
     Y:JN = 1: GOTO 708
752  IF V$ = "N" THEN 708
754  IF V$ = "E" THEN 702
756  GOTO 744
758  HPLOT XA,YA TO X,Y: GOTO 730
900  TEXT : PRINT "ARRAY READER": FOR T = 1 TO 3000: NEXT
910  FOR N = 0 TO 100
915  PRINT N","A%(0,N) / 10","A%(1,N)","A%(2,N)","A%(3,N)
920  NEXT N: FOR T = 0 TO 3000: NEXT : GOTO 300


CRUN WIDE PR¬\
LOAD WIDE PRINTER
RUN
DATA PRINTER FOR 'STARGATE 100'.
RRAY LOADING ...
TRINGS LOADING ...

ORMAT IS :-
A$(N), A%(0,N), A%(1,N), A%(2,N), A%(3,N)
WHERE A%(0,N) = INT( 10 * ABS'MAGNITUDE OF STAR.)

ACKNOWLEDGEMENTS TO :-

LLEN'S ASTROPHYSICAL QUANTITIES.  (ATHLONE PRESS, LONDON,WC1B-3PY)

STRONOMY DATA BOOK. (D&C(HOLDINGS) LTD, NEWTON ABBOT)

UINNESS BOOK OF ASTRONOMY. (GUINNESS SUPERLATIVES,ENFIELD,EN2-6DJ)

10 SEC. DELAY)
```

```
)  SOL G2V    48, 0, 0, 0
1  -37'15492 M4V    86, 11, 0, -9
2  +43'44 M1V+M6V    103, 8, 1, 8
3  BETA HYI G1 IV    37, 4, 0, -20
4  ETA CAS GOV+MOV    45, 10, 2, 16
5  VAN MAANEN'S DG    143, 13, 3, 1
6  L726.8 M5+M6    153, 8, 3, -3
7  TAU CETI G8 VI    57, 10, 5, -3
8  L1159-16 M8    139, 13, 7, 3
9  82 ERI G5    52, 10, 11, -14
10  EPS ERI K2V    61, 6, 8, -2
11  OMICRON (40) ERI K1V+DA+M4    59, 7, 14, -2
12  KAPTEYN'S MO    108, 2, 9, -9
13  -3'1123 M1V    91, 3, 19, -1
14  ROSS 47 M6VI    127, 2, 19, 4
15  LP 658-2 DK    156, 1, 20, -1
16  -21'1377 M1V    93, -1, 17, -7
17  ROSS 614 M7+M    131, -2, 13, -1
18  SIRIUS A1V+DA    14, -2, 8, -2
19  WOLF 294 M4    113, -4, 16, 11
20  ROSS 986 M5    126, -4, 14, 12
21  LUYTEN'S 1668 M5    119, -4, 11, 1
22  PROCYON F5V+DF    26, -5, 10, 1
23  ROSS 882 M4    122, -9, 18, 1
24  L97-12 D    155, -3, 6, -17
25  L674-15 M    150, -10, 15, -7
26  +53'1320 MOV+MOV    87, -9, 8, 15
27  +50'1725 MOV    83, -9, 5, 11
28  +20'2465 M4V    109, -14, 7, 6
29  WOLF 359 M8    166, -7, 2, 1
30  LL 21185 M2V    104, -6, 2, 5
31  +44'2051 M2V+M8    101, -12, 3, 12
32  L145-141 DA    130, -7, 0, -14
33  AC+79'3888 M4 VI    123, -3, 0, 16
34  ROSS 128 M5    135, -11, 1, 0
35  WOLF 424 M6+M7    149, -14, -2, 2
36  LL25372 M4V    100, -14, -7, 4
37  PROXIMA (ALPHA'C') CEN M5    154, -2, -1, -4
38  -11'3759 M4    123, -16, -12, -4
39  ALPHA CEN (AB) G2V+K5V    43, -2, -1, -4
40  -20'4125 K5V+M2V    70, -12, -12, -7
41  -40'9712 M4    111, -9, -12, -13
42  -12'4523 M5    121, -5, -12, -3
43  WOLF 629 M4VI+M4+M5+M    127, -6, -19, -3
44  +45'2505 M3+M4    109, -3, -14, 15
45  36 OPH K1V+K1V+K5V    63, -3, -16, -8
46  -46'11540 M4    110, -2, -10, -11
47  -44'11909 M5    128, -1, -11, -11
48  +68'946 M4V    107, -1, -6, 15
49  L205-128 M    140, -1, -10, -16
50  BARNARD'S M5V (2 PROBABLE PLANETS)    132, 0, -6, 0
51  70 OPH KOV+K5V    56, 0, -17, 1
52  STRUVE 2398 M4+M5    111, 1, -6, 10
53  ROSS 154 M4    133, 2, -8, -4
54  +4'4048 M4V+M5    103, 6, -18, 2
55  L347-14 M7    148, 4, -12, -13
```

```
56  SIGMA DRA K0V     59, 3, -6, 17
57  ALTAIR A7V     22, 7, -15, 3
58  SIGMA PAV G6V     47, 4, -6, -17
59  -36'13940 K3V+M5     65, 8, -13, -11
60  -45'13677 M0V     90, 8, -12, -14
61  61 CYGNI K5V+K7V     75, 6, -6, 7
62  -39'14192 M0V     87, 7, -6, -8
63  -49'13515 M1V     103, 8, -6, -12
64  EPSILON INDI K5V     70, 5, -3, -9
65  KRUGER 60 M3+M4     118, 6, -3, 11
66  L789.6 M7     146, 10, -4, -3
67  +43'4305 M4     116, 11, -4, 12
68  ROSS 780 M5     117, 15, -5, -4
69  -36'15693 M2V     95, 9, -2, -7
70  +19'5116 M4V+M6     113, 20, -3, 7
71  ROSS 248 M6     148, 7, -1, 7
72  1'4774 M2V     101, 19, -1, 1
73  ALPHA HYI F0V     29, 12, 7, -28
74  'PI'^3 ORI F6V     37, 7, 24, 3
75  ALPHA AUR G8III+F CAPELLA     -6, 6, 30, 32
76  XI GEM F5 IV     19, -12, 61, 14
77  ALPHA PIC A5V     21, -6, 26, -51
78  ALPHA GEM ABC A1V+A5+M0V CASTOR     -1, -16, 35, 23
79  BETA GEM K0 III POLLUX     10, -14, 27, 16
80  IOTA UMA ABC A7V+M1V+M     22, -23, 23, 36
81  BETA LEO A3V     15, -42, 2, 10
82  GAMMA VIR AB F0V+F0V     35, -32, -6, -1
83  ETA BOO G0 IV     27, -27, -15, 10
84  ALPHA BOO K1 III ARCTURUS     -3, -29, -19, 11
85  ZETA HER AB G0 IV+K0V     31, -9, -24, 15
86  ALPHA LYR A0V VEGA     5, 3, -21, 16
87  ETA CEP K0 IV     27, 14, -17, 40
88  ALPHA PSA A3V FOMALHAUT     20, 18, -6, -12
89  BETA AQL G8 IV+M3V     33, 20, -37, 4
90  CHI DRA F7V     41, 0, -8, 24
91  DELTA ERI K0 IV     40, 16, 23, -5
92  GAMMA LEP ABC F6V+K2V+M     41. 1, 24, -11
93  IOTA PEG F5V     34, 33, -19, 17
94  XI UMA G0V+G0V     45, -21, 4, 13
95  10 UMA F5V+K     34, -24, 23, 30
96  PSI VEL F2 IV+K     27, -31, 23, -33
97  BETA VIR F8V     38, -32, 1, 1
98  ALPHA FOR F8 IV+K5     31, 26, 29, -22
99  KAPPA PHE A7V     30, 35, 4, -35
100  ETA LEP F0V     28, 0, 48, -13

DONE !!
```



SHARP POINT

by A.Tiplady
ARIC, C ENG, MI CHEM E.

THIS is the first of a series of occasional articles dealing the the SHARP MZ80K personal computer. Since its introduction, last, autumn, the MZ80K has established itself firmly in the home and small business computer market although, as yet, it does not have the software and technical backup of the older machines.

As most owners will know SHARP are a bit tight fisted with information on the MZ80K. This is a shame since, in a straight contest, the MZ80K would leave most home computers standing. Perhaps SHARP believe that we will appreciate it more if we have to work hard for it.

For those who do not believe in hair shirts the following bits and pieces may be useful.

POKE 10167,1 will allow peeking of any memory location, including the monitor and basic, when Basic SP 5025 is loaded. The pioneers, with Basic SP 5010, will find POKE 10404,1 disables the PEEK protection. When peeking into the monitor and basic, print CHR$ of the PEEK so that the word lists can be readily recognised. SP 5010 may be slow but at least it will run the 3-D maze without dropping out on 16 GOSUB at line 7740. To run the maze on SP 5025 change the GOSUB at 7740 to GOTO 3000 and the four RETURNS in the subroutine to GOTO 7750.

Whilst on the subject of SHARP's shop demonstration programmes I would love to know who modelled for SPOCK in their STARTREK. Incidentally SPOCK is federation qualified as a starship first officer but is neither a medic. nor a PhD. Dr Spock is the American child-care expert.

To save typing in all those reverse field arrows cursor movement can be controlled by POKE 4466,X. Values of X from 0 to 24 moves the cursor to the start of each of the 25 screen lines.

Maybe these few SHARP POINTS will start the information on the MZ80K circulating and let us all make fuller use of this very powerful computer. Anyone with useful points should send them in to the editor so he can spread the word.

CRYSTAL ELECTRONICS
CC ELECTRONICS

XTAL Basic 2·2
NOW ON SHARP MZ80K

All of the features of SHARP BASIC and more. Occupies 5K less memory, thus effectively increasing memory size for programs.

MZ80K 20K RAM £520+VAT
(with XTAL BASIC leaves 11K for programs)
XTAL BASIC for SHARP £40+VAT

Coming Soon—PETSOFT* programs in XTAL BASIC format for SHARP & NASCOM—Prices as PETSOFT list+20%.

NASCOM 1 & 2 owners—XTAL BASIC 2.2 £35+VAT
NEW EPROM version £100+VAT
(please state monitor used)
EPROM version runs in EOOOH-FFFFH.

APPLE II PLUS OWNERS
APPLE INTEGER BASIC on disc for 32K or 48K APPLE.
Now you can run nearly *all* of your programs—
£20+VAT

** PETSOFT is a tradesmark of ACT PETSOFT LTD.*
Members of Computer Retailers Association & Apple Dealers Association

Shop open 0930-1730 except Saturday & Sunday
40 Magdalene Road, Torquay, Devon, England. Tel: 0803 22699

Access and Barclaycard welcome.

COMPUTERS AND COMPONENTS

The multi-minded micro

The Sharp MZ-80K with BASIC, Machine Language & Assembler.

Sharp's microcomputer has three languages. There's BASIC of course and now tapes and manuals for Machine Language & Assembler Programming. Trust Sharp, leaders in the field of electronic business equipment to come up with such a multi-minded microcomputer.

- ❊ Z-80 based CPU ❊ 4K bytes monitor ROM
- ❊ Internal memory expansion up to 48K bytes of RAM
- ❊ 14K extended BASIC (occupies 14K bytes of RAM)
- ❊ 10" video display unit – 40 characters x 25 lines
- ❊ 80 x 50 high resolution graphics
- ❊ 78 key ASC11 keyboard alphabet (capital and small) plus graphics ❊ Built in music function
- ❊ Fast reliable cassette with tape counter – 1200 bits/sec
- ❊ 50 pin universal BUS connector for system expansion – printers, floppy discs, etc.

Insist on SHARP

Audio, Video, Business Systems, Calculators, Cash Registers, Copiers, Microwave Ovens.

Sharp Electronics (UK) Limited, Sharp House, Thorp Road, Newton Heath, Manchester. M10 9BE Tel: 061-205 2333

Find out more about the multi-minded micro by filling in this coupon right away.

I would like to know more about the **SHARP MZ-80K** and the address of my nearest Specialist.

NAME LSG/5

ADDRESS

TEL:

Sharp Electronics (UK) Limited, Sharp House, Thorp Road, Newton Heath, Manchester. M10 9BE Tel: 061-205 2333

LETTERS TO THE EDITOR

65 Blodes Street,
Lancaster,
Lancs. LA1 1TS

22nd March 1980

Dear Sir,

I enclose a listing of a very useful program for a Nascom 1/2 that has Nas-sys. It is a "modify" version of the "dis-assemble" INTAB that appeared in Computing Today. It is located from 3000, but can easily be relocated. The start address of the program to be examined must follow the address of INTAB in the EXECUTE COMMAND - see example with print out.

If a program is to be modified then touching M instead of the space bar will call the Nas-sys modify routine. When the mods have been carried out a normal finish to the modify i.e. followed by enter returns one to INTAB where space bar depressions allow you to continue through the program.

I have found it very useful and am sure other people will.

Yours sincerely,
W.S. LOUNDS

```
T3000  30 E8
 3000  ED  5B  0E  0C  CD  D0  30  01  60
 3008  05  00  DF  6A  7A  DF  68  00  47
 3010  7B  DF  68  00  EF  20  20  00  31
 3018  1A  FE  EF  28  58  FE  DD  28  D2
 3020  23  FE  FD  28  1F  FE  ED  28  C8
 3028  32  0E  1A  21  84  30  ED  81  25
 3030  06  03  28  08  01  1A  00  ED  A4
 3038  81  06  02  28  02  06  01  CD  1F
 3040  6A  30  18  CD  CD  68  30  21  68
 3048  9B  30  ED  B1  28  E2  21  BD  C9
 3050  30  0E  0B  ED  81  28  E6  06  7B
 3058  02  18  E4  CD  68  30  0C  21  18
 3060  88  30  ED  B1  28  CA  18  D5  F5
 3068  06  01  DF  68  00  DF  69  00  2E
 3070  13  1A  10  F6  C9  CD  68  30  01
 3078  A7  28  C2  F6  00  F7  00  00  26
 3080  13  1A  18  F4  01  11  31  32  5E
 3088  3A  C2  C3  C4  CA  CC  CD  D2  70
 3090  D4  DA  DC  E2  E4  EA  EC  F2  D8
 3098  F4  FA  FC  21  22  2A  36  CB  20
 30A0  06  0E  10  16  18  1E  20  26  86
 30A8  28  2E  30  38  3E  C6  CE  D3  3B
 30B0  D6  D8  DE  E6  EE  F6  FE  DF  16
 30B8  43  48  53  5B  73  7B  09  19  34
 30C0  23  29  2B  39  E1  E3  E5  E9  32
 30C8  F9  E7  E7  E7  E7  E7  E7  E7  42
 30D0  CF  FE  4D  28  01  C9  E5  D5  C6
 30D8  EB  2B  22  0C  Oc  DF  6A  CD  6E
 30E0  40  02  D1  E1  18  EA  E7  E7  D4

E 3000  3000

 3000  ED  5B  0E  0C
 3004  CD  D0  30
 3007  01  05  00
 300A  DF  6A
 300C  7A
 300D  DF  68
 300F  00
 3010  7B
 3011  DF  68
 3013  00
 3014  EF      00
 3018  1A

E3000  0

 0000  31  00  10
 0003  D7
 0004  08
 0005  C3  82  03
 0008  DF  62
 000A  D8
 000B  18  FB
 000D  C3  9A  03

E3000  350

 0350  38  0B
 0352  D6  07
 0354  FE  0A
 0356  D8
 0357  FE  10
 0359  38  02
 035B  37
 035C  C9
 035D  13
 035E  34
 035F  23
 0360  ED  6F
 0362  23
 0363  ED  6F
 0365  2B
```

Dear Sir,

We have just obtained the early copies of the Gazette through a 'backdated' order and I would like to compliment you on the level of your articles and the range of interests you cover, which suits my needs perfectly.

In the January issue, I note the entry 8K BASIC Bug? in the NASCOM Notes and assume I am not the only reader to attribute the apparent misbehaviour of the routine to the insertion to the print buffer of a new line. character after 47 consecutive characters. This facility is sometimes of dubious value and we use POKE 4162,255 to suppress it (POKE 4162,47 restores it), while POKE 4162.0 gives automatic double line spacing on the screen if required.

Yours sincerely,
Gwyneth Pettit (Mrs.),
Computer Unit,
Leeds Polythechnic.

Dear Sir,

I organize the library of the IMMC and several of our members who regularly buy your Software Gazette have expressed interest in the M5 language and a disassembled testing of M5 has been submitted for inclusion in our library.

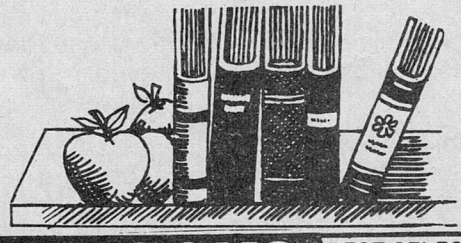
Would you be willing to let us put the source of M5 (corresponding to the Hex dump in Gazette No.1) into the IMMC library - with appropriate acknowledgements?

The first two copies of the Gazette have been very much appreciated for all the information they provide on a wide range of micros. I personally found them very helpful.

Yours faithfully,
Derek Brough,
Internation Nascom Microcomputer Club.

Editors reply:

The purpose of the Gazette is the spread of microcomputer software, our reprints policy as stated at the front of the magazine is as liberal as possible. We therefore welcome organisations like the IMMC spreading our software.



MICRODIGITAL BOOKS

BEST SELECTION-BEST PRICES-BEST SERVICE
25 Brunswick Street, Liverpool 2. Tel: 051-236 0707 (Mail Order) 051-227 2535 (All Other Depts)

Accounts Payable & Receivable	£9.95
Active filter Cookbook	£10.90
Adaptize Info, Processing	£8.75
Advanced Basic	£4.00
Algorithms & Data Structure Equals Programs	£14.00
Anatomy of a Compiler	£15.00
Apl. — An Interactive Approach	£9.50
Artificial Intelligence	£12.00
Artist & Computer	£3.96
Art Computer Programming Vol. 1.	£9.50 S/B £16.45 H/B
Art Computer Programming Vol. 2.	£16.45
Art Computer Programming Vol. 3.	£16.45
Assembley Level Programming for small computers	£12.76
Analysis and design of Digital Circuits and Computer Systems	£16.40
Accent on Basic	£4.95
About Computers	£6.95
Active Filters	£6.45
Analog/Digital Experiments	£7.15
A guided tour of Computer Programming in Basic.	£4.16
A quick look at Basic	£4.45
Aple II Operators Manual	£5.50
Apple II Integer Basic manual	£3.55
Apple II Applesoft Extended Basic Manual	£5.75
Advanced Business, Billing, Inventory, Investments, Payroll	£26.95
An Introduction to Your new Pet.	£1.00
Basex	£5.60
Basic A Hands on Method	£6.50
Basic and the Personal Computer	£10.00
Basic, Basic.	£4.80
Basic Computer Games Micro.	£5.50
Basic from the Ground Up	£7.00
Basic Microprocessors and the 6800.	£7.20
Basic with Business Application	£8.40
Basic with style/Programming proverbs.	£3.60
Basic WorkBook	£3.70
Best of Creative Computing Vol. I.	£6.95
Best of Creative Computing Vol. II.	£6.95
Byte Book of Computer Music	£7.00
Beginners Glossary and Guide	£5.75
Beginners Basic	£7.50
Byte Vol I	£8.45
Basic — a unit for Secondary Schools	£4.45
Basic Programming	£6.95
Basic Primer	£6.95
Collection of Programming Problems and Techniques	£12.00
Computer Crime	£12.00
Computer Data Directory	£3.98
Computer Lib/Dream machines	£5.95
Computer Models of Thought and Language	£17.00
Computer Power and Human reasons	£4.76
Computer resourse book Algebra	£4.00
Computer Science a First Course	£15.00
Computer Science Programming in Fortran IV	£7.80
Computer/Science Projects and Study problems	£8.75
Concurrent Pascal Compiler	£6.40
Conference Proceedings of the 1st West Coast Computer Fair	£9.56
Conference Proceedings of the 2nd West Coast Computer Fair	£9.56
Conference Proceedings of the 3rd West Coast Computer Fair	£9.58
Consumer Guide-Personal Computers & Micros.	£5.00
Content Addressable Parallel Processors.	£11.20
Computer Dictionary (sams)	£6.95
Computer Dictionary & Handbook	£11.95
Computer & Program Guide for Engineers.	£9.45
Computers for the Physicians Office	£15.50
Computerisation for small Business	£9.95
Computer Quiz Book	£5.45
Computer Programs that Work.	£2.95
Computer Music	£7.00
Computer Data Directory	£3.98
Computer Rage (a board game)	£6.95
Clarcia's Circuit Cellar	£5.60
Designing with TIL Circuits	£24.80
Design of Well Structured Prog's	£10.00
Dictionary of Microcomputing.	£10.00
Digital Computer Fundamentals.	£14.25
Dr. Dobbs Jounral Vol. 1.	£10.00
Digital IC Equivalents & Pin Connections.	£1.25
Designing M/Computer Systems	£5.40
Editor/Assembler Systems for 8080/8085 Based Systems	£11.96
8080 Programmer Pocket guide	£1.95
8080A Bugbook	£7.65
8080 Programming for Logic Design	£5.95
8080A/8085 Assembler Language Programming.	£6.95
8080/8085 Software Design	£7.65
8080 Machine Language Programming for Beginners	£5.10
8080 Microcomputer Experiments	£10.25
8080 Standard Monitor	£9.95
8080 Standard Editor	£9.95
8080 Standard Assembler	£9.95
57 Practical Programs Basic	£6.40
First Book of Kim	£7.00
Fortran Coloring Book	£5.56
Fundamentals of Data Structure	£15.00
Fundamentals of Computer Algorithms	£15.00
Fundamentals and Application of Digital Logic	£6.00
From the Counter — Bottom Line	£8.75
Fundamentals of Digital Comps	£7.25
Fortran Programming	£6.75
Fortran Workbook	£4.75
Fortran Fundamentals	£3.45
Fun with Computers & Basic	£5.45
555 timer	£5.35
50 Circuits using 7400 Series IC's	75p
Fortran Fundamentals a short course	£2.95
Game Playing with Basic	£5.50
Game Playing with Computers	£10.20
General Ledger	£9.95
Getting Involved with Your Own Computer	£4.75
Getting Acquainted with Micros	£7.95
Guide to Scamp Programming	£4.00
Games with a Pocket Calculator	£1.75
Games, Tricks & Puzzles for a Hand Calculator	£2.49
How to Build A Computer Controlled Robot.	£5.95
How To Profit from Your Personal Computer	£5.50
How To Program Microcomputers	£7.50
How You Can Learn To Live With Computers	£7.00
How to buy/use Mini-Micros	£7.50
How to Build a Working Digital Computer	£4.60
How To Package Your Software	£27.50
Home computer Revolution	£2.75
Home Computers: A Beginners Glossary and Guide.	£4.95
Hobby Computers Are Here	£3.95
Incredible Secret Money Machine.	£4.75
Interface Circuits Data Book	£3.50
Intro to Artificial Intelligence	£14.80
Intro to Computer Prog	£4.75
INTRODUCTION TO MICROCOMPUTERS. Vol 0 £5.95 Vol II £18.95 Vol I £5.95 Vol III £11.95	
Intro to Personal & Business Computing	£4.95
Intro to Prog. Prob. Solving with Pascal. Schneidser Et Al	£8.50
Instant Basic	£7.20
Introduction to Basic	£7.15
Interfacing and Communications	£4.95
Introductory Experiments in Digital Electronics Vol I	£8.75
Introductory Experiments in Digital Electronics Vol 2	£7.75
Introduction to TRS — 80 Graphics	£5.25
Instructor's manual for Fortran Programming	£6.95
Illustrating Basic	£2.25
IC Optamp Cookbook	£8.95
Introduction to Microprocessors & Computing	£2.40
Kim 1 Users Manual	£5.00
Linear Control Circuits Data Book	£4.00
Link 68 — M6800 Linking Loader	£5.50
Little Book of Basic Style	£4.75
Linear IC Principles Experiments	£7.16
Math Elements for Computer Graphics	£10.40
Microcomputer — Based Design	£8.00
Microcomputer Handbook	£15.96
Microcomputer Primer	£6.35
Microelectronics	£3.25
Micro Problems Solving Using Pascal	£7.50
Microprocessor Interfacing techniques	£7.95
Microprocessor Lexicon	£2.00
Microprocessor System Design	£14.00
Microprocessor From Chips to Systems	£7.50
Mind Appliance	£4.75
Modern Operational circuit Design	£18.60
Mondeb M6800 Monitor/debug	£3.50
Microcomputers at A Glance	£6.45
Micro/Computer Interface with the 8255 PPI Chip	£10.00
Microformula (TM)	£2.95
Microsoft (TM) Basic	£7.00
Microprocessor Basics	£7.60
Micros — New Directions for Designers	£7.60
Microprocessors Data Manual	£5.50
Micros for Business Applications	£7.15
More Basic Computer games	£5.50
Microcomputer Potpourri	£1.75
Modern Operational Circuit Design	£18.60
Microprocessor Basics	£7.60
Microprogrammed Apl Implementation	£14.75
Mos Memory Data Book	£2.80
Microprocessor Encyclopædia Vol 2.	£7.45
9900 Family systems Design & Data Book	£7.95
1975 U.S. Comp Chess	£4.75
1976 U.S. Comp Chess	£4.75
NCR Basic Electronics	£7.15
NCR Data Communications	£7.15
NCR Data Processing	£6.35
Nascom 1 — Hardware Notes	£1.50
Nascom 1 — Seminar Notes	£1.50
Optoelectronics Data Book	£3.50
Osbourne Updated Subscriptions	
Vol. 2. (6 issues)	£18.95
Vol. 3. (6 issues)	£18.95
Vol. 2 & 3. combined	£30.00
Binders 2 or 3	£5.75
Pascal User Manual & Report	£5.25
Payroll with cost accounting	£12.00
Power Semi-conductor Data Book	£7.50
The Mighty Micro	£5.50
Z — 80 Programming Manual	£4.50
TTL Databook	£7.00
TIS Workbooks (1-6) each	£3.00
TI S 6800 Tracer	£4.80
TTL Cookbook	£6.95
Peanut & Jelly Guide to Computing	£6.45
The Best of Micro	£5.50
Z — 80 and 8080 Assembly Prog.	£4.80
6500 Programming manual	£10.50
6500 Hardware Manual	£10.50
Software Gazette	50p

WELCOME
Prices include Postage and Packing anywhere in the UK.

COMPUTER RETAILERS ASSOCIATION

The Computer Retailers Association is a voluntary organisation of leading micro-computer dealers and service organisations.
The purpose of the Association is to maintain and improve standards of trading and customer support within the industry and to present the industry's case to the outside world. The Association also acts as a forum where members can discuss common problems.

Membership is open to all reputable organisations within the industry who are willing to subscribe to the Association's Code of Practice and who meet the Conditions of Membership.

Founder Members of the Association include:

Camden Electronics
Comart Ltd.
Crystal Electronics
Datron Interform Ltd.
Equinox Computer Systems Ltd.
Great Northern Computer Services Ltd.
H.B. Computers Ltd.
Intex Datalog Ltd.
Isher - Woods Business Systems
Keen Computers Ltd.
Microdigital Ltd.
Newbear Computing Store
Petalect Electronics Ltd.
Sheffield Computer Centre
Sintrom Microshop
Tridata Micros Ltd.
Xitan Systems

Conditions of Membership

Members must satisfy the Membership Committee that they will comply with the Code of Practice and that they meet the Conditions of Membership.

These Conditions include;

i Members must have a significant interest in supplying micro-computers or related products and services to end-users.

ii Members must have a permanent display area where their products and services can be effectively demonstrated.

iii Members must have a telephone, permanently manned during advertised office hours.

iv Members must have adequate experience in both hardware and software to properly advise prospective customers on their products and services and to give effective after-sales support.

v Members must be committed to the spirit of the Association and must comply with the Association's Code of Practice.

Code of Practice

The Code of Practice aims to ensure that members always conduct their business professionally and ethically.

Amongst the clauses in the code are the following;

i Members will not mis-represent their experience or capabilities and will not knowingly trade upon the ignorance of prospective customers.

ii Members will not deliberately advertise goods for sale which are not currently available. Members will not make misleading claims in their advertisements and will avoid terms that are likely to be misunderstood.

iii Members will acknowledge orders with cash received through the mail within seven days and will give customers the right to cancel and have a full refund if the order cannot be filled within thirty days.

iv Members will tell prospective customers of any vested interests they have when recommending goods or services.

v Members will treat as confidential information learnt about the customer's business.

vi Members agree to submit any disputes between themselves and customers to arbitration by a Board set-up by the Association and will be bound by its findings.

What CRA means to Dealers

The CRA is the only body fully representing the micro-computer industry. All organisations with something to say about the industry; about computers in education; about trading standards; about government policy; can join the Association and get the voice of the whole industry speaking on their behalf.

Not every organisation in the industry is suitable to be a member, but those that are good enough to belong will want the world to know it. Membership of the CRA is a public recognition that they are capable, professional and ethical.

The CRA has a full-time secretariat. The CRA advertises its aims and membership in the computer press. The CRA issues an internal Handbook for members and plans to publish an annual Directory of members, their products and their services for sale to the public. The CRA meets regularly to discuss important issues facing members and the CRA provides an information exchange. All these services are available to members.

What the CRA means to Users

Using micro-computers has been likened to a game of snakes and ladders where there are "more snakes than ladders"*. The best way to avoid these snakes is to deal with a reputable organisation that will help users to avoid them and will give proper after sales support. Members of the CRA are, by definition, just such organisations.

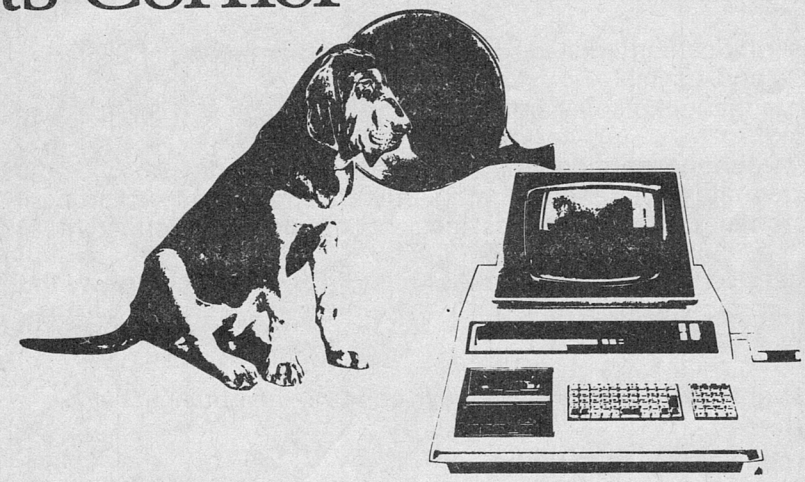
All users should support members of the CRA, because when they need help, CRA members will support them.

For further details please contact;

Mrs H.M.W Gibbons, Assistant Secretary, Computer Retailers Association
Owles Hall, Buntingford, Herts. Tel: Royston (0763) 71209

* *"Filling you in on small business systems" Computer Management - October 1979*

Pets Corner



J. Stout

Department of Architecture
Liverpool Polytechnic
53 Victoria Street
Liverpool
L1 6EY

INDENTATION

To illustrate the extent of loops in a BASIC program it is helpful to indent those lines of the program which lie within a FOR...NEXT statement, e.g.

```
10 FOR I=1 TO N
20 FOR J=1 TO N
30 A(I,J)=0
40 NEXT J
50 NEXT I
```

If you try this on a PET the leading spaces between the line number and the first non-space character of the line will be removed, with the intention of saving space presumably. One alternative is to replace the spaces with colons, as in Dr Beynon's program on page 61 of the last issue.

On the PET the following technique may be used, not only to generate the indentation but also to create blank lines if necessary. Precede the leading spaces with the shifted & character (which prints as a checkerboard character). This seems to protect the leading spaces from being removed. If a blank line is to be created simply terminate the required number of spaces with another shifted &. No untoward effects have been noticed with this technique, and if the resulting program is examined useing the ROM based monitor, then the shifted & characters are not stored but the spaces are.

My thanks to Mark Atherton of Merchant Taylor's School for this information.

SPEEDING UP PROGRAMS RUNNING WITH THE COMPU/THINK DISK.

If you are lucky enough to own a COMPU/THINK disk it is interesing to try the following experiment.

Set up a simple loop, e.g.

```
10 FOR I+1 TO 500
20A=A+1
30B=B+1
40 NEXT I
```

running first in a PET which has just been switched on, and then in the same PET **after** the COMPU/THINK Disk Operating System has been switched on (with the SYS(11*4096) command). If you time the program runs you should notice that the second run takes noticeably longer, usually in the region of 20%-30% longer. The reason for this is that the PET accesses the next character of the BASIC program it is executing through a machine-code subroutine in the first page of RAM. This routine's job is to ignore spaces, return with the carry set if the character retrieved was a digit, with a zero result if the character was end of line or statement together with various other bits of information which are not obvious from the coding (reproduced below). The importance of this routine is that it is the only way in which an additional routine, written by a user, can examine characters of the BASIC program and look for commands to be executed by the additional routine, rather than by the BASIC as it stands. Thus any subsystem which adds commands to PET BASIC, e.g. the Programmer's Toolkit, the Commodore DOS support package or the COMPU/THINK Disk Operating System must alter this subroutine, usually by adding a JMP instruction to the additional routines which scan the character and any other data needed (e.g. is the character at the start of a line, is it in direct mode and so on), decide whether one of the additional commands is being called and if so execute it, otherwise continue with the execution of the BASIC program normally. This of course means additional work being done by every call to the page zero routine, and since this is called for every character in the program being executed the overhead can mount up, especially within loops.

If you have a program which does large amounts of processing between calls to the operating system, e.g.

in a Word Processor, it is well worth adding the little section of programming below, and calling the subroutine at 60000 at the start of the program, then to shut the Disk Operating System off execute SYS(826), while to switch it back on execute SYS(11*4096). Since the SYS calls themselves take up very little time sprinkle them liberally about the program and see what effect they have. One word of warning: always make sure that when the program is stopped, either because it has finished or because you press STOP, that you enable the system, otherwise when you try to save, load or do anything with the disk you will get ?SYNTAX ERROR, having removed the link between BASIC and the Operating System.

```
60000 REM POKE MACHINE CODE SUBROUTINE INTO SECOND CASSETTE BUFFER
60010 REM FOR DISABLING THE COMPU/THINK DISK OPERATING SYSTEM.
60020 FOR I=1 TO 13
60030   READ M:POKE 825+I,M
60040 NEXT I
60050 RETURN:REM WITH ROUTINE SET UP.
60060 DATA 169,230,133,112,169,119,133,113,169,208,133,114,96
60070 REM IF YOUR PROGRAM HAS ITS OWN DATA LINE 60060 MAY HAVE TO BE MOVED.
```

The machine code program is as follows:

```
033A A9 E6      LDA   #$E6      ;AN INCREMENT PAGE ZERO INSTRUCTION.
033C 85 70      STA   $70       ;FIRST BYTE OF THE PET'S ROUTINE.
033E A9 77      LDA   #$77      ;THE ADDRESS OF THE BYTE IN PAGE ZERO
                                ;TO BE INCREMENTED.
0340 85 71      STA   $71       ;SECOND BYTE OF ROUTINE.
0342 A9 D0      LDA   #$D0      ;A BRANCH IF NOT EQUAL INSTRUCTION
0344 85 72      STA   $72       ;THIRD BYTE OF THE ROUTINE.
0346 60         RTS             ;RETURN TO BASIC.
```

The above example is for the new ROMS. For the old ROMs the data line becomes:
60060 DATA 169,230,133,194,169,201,133,195,169, 208,133,196,96 the page zero routine starting at $C2 rather than $70, and the byte incremented being at $c9 rather than $77. The enabling and disabling calls remain the same. If you use the first 13 bytes of the second cassette buffer for some other purpose the routine POKED in there is completely relocatable, the only changes to be made will be to alter the address 825 in the line 60030 to one before the first free address, while the disabling call will be to the first byte free.

The page zero routine which is overwritten when the Disk Operating System is set up is as follows:

```
0070 E6 77      INC   $77     ;INCREMENT LOW O RDER BYTE OF LDA INSTRUCTION.
0072 D0 02      BNE   +$02    ;IF RESULT WAS NOT ZERO DON'T INCREMENT
                              ;HIGH ORDER.
0074 E6 78      INC   $78     ;INCREMENT HIGH ORDER, I.E. PERFORM A 16-BIT
                              ;INCREMENT.
0076 AD -- --   LDA   $----   ;ADDRESS DEPENDS ON WHETHER DATA COMING FROM
                              ;PROGRAM, INPUT BUFFER ETC.
0079 C9 3A      CMP   #$3A    ;ASCII CODE FOR COLON.
007B B0 0A      BCS   +$0A    ;IF CHARACTER GREATER THAN OR EQUAL TO :
007D C9 20      CMP   #$20    ;ASCII CODE FOR SPACE.
007F F0 EF      BEQ   -$11    ;SKIP SPACES.
0081 38         SEC
0082 E9 30      SBC   #$30
0084 38         SEC
0085 E9 D0      SBC   #$D0
0087 60         RTS
```

SORTING

As a companion piece to last issue's discussion of searching this issue looks at the associated subject of sorting. The two processes can be quite intimately connected, for example we saw in the last issue that a very powerful searching method (the binary search) was possible only if the array we were searching was ordered. Similarly in one of the searching methods proposed here the process of searching becomes important.

Two issues cannot hope to cover these subjects to any great depth, so that anyone wanting to find more detail should consult one or both of the following books:

Knuth, D.E. Sorting and Searching, The Art of Computer Programming, vol. 3. Adisson-Wesley, 1973.

Writh, N., Algorithms + Data Structures = Programs Prentice-Hall, 1976

Knuth's book is **the** reference, but it is heavily mathematical and the programs in it are written in an assembly language designed for a hypothetical machine called MIX. The second ot the two books has been mentioned before in this column, and while it does not cover sorting to any great depth (the book contains 5 chapters in 350 pages as opposed to Knuth's 2 chapters in 560 pages) the programs in it are written in Pascal and the entire book is much easier to read. Much of this section is in fact borrowed from Writh's chapter on Sorting.

As in last issue we start this discussion with a description of the problem we wish to solve. We will be given an array FILE (), with NE number of elements in the array locations FILE (1), FILE (2), ... , FILE (NE). Before the program is run we know nothing about the relative order of any two elements in the array, but after the program has been run we require the array to be sorted, so that:

FILE (1)«FILE (2)«FILE (3)«...«FILE (NE) (note that we ignore the zero'th element of the array)

Wirth classifies sorting methods into straight and advanced methods, the straight methods requiring a computational effort in the order of NE^2, while the advanced methods usually require an effort in the order or NE*LOG(NE). Of the four sorting methods illustrated here three belong to the straight categary and 1, QUICKSORT, belongs to the advanced category.

For each sorting method a program to implement the sort will be given, together with a brief explanation of the method behind the program. Finally a comparison of the times taken by the four methods will be given.

Sorting by Insertion [Listing 1].

IN this method the array is divided into two sections, with the index I marking the boundary between them. All elements to the 'left' of I are sorted (initially only one element is to the 'left' of I when it is equal to 2) and at each step the element pointed to by I is inserted into the sorted sequence in its correct location. I is then incremented by one and the process repeats until there are no more elements to the 'right' of I.

Sorting by Insertion with Sentinel [Listing 2].

IN much the same way as we simplified some of the searching algorithms by setting the zero'th element of the array to the value searched for, the selection by insertion can be simplified by setting the zero'th element of the array to the value of the I'th element. This means that the resulting program is somewhat shorter and more elegant, but as we shall see takes slightly longer to run. Note that at some time the text in line 100 **must** fail since, when J=0, X=FILE (J).

Sorting by Selection [Listing 3]
In this method we use the same idea of dividing the array into two sequences, using I as the boundary pointer, but in this method the unsorted sequence is searched for the smallest remaining element and this is placed at the rightmost end of the sorted sequence. Working out the theoretical values of the timing of the selection sort gives it an advantage over the insertion sort and as we shall see the experimental results bear this out.

Sorting by Exchange [Listing 4].

Also known as Bubblesort this is one of the best known sorting methods, probably because of its name, since its performance is not at all memorable. In each pass through the I loop the lowest of the elements FILE(NE) to FILE (I) is placed in FILE(I-1), i.e., the higher elements bubble up through the array, this second way of looking at the process no doubt being where the algorithm obtained its name. To quote Wirth directly 'the Bubblesort has hardly anything to recommend it except its catchy name'.

Variations on the Bubblesort include using a flag to indicate whether any exchanges have been made on the last pass through the inner loop. If none have been made then obviously the array is sorted, since every pair of elements in the unsorted sequence (from J=NE to I) satisfies the relation FILE (K) FILE(L) where K is less than L. A version called Shakersort, which does not only bubble up elements but also sinks elements alternately, produces only a minor improvement, unless the original sequence is known to be almost in order.

Quicksort [Listing 5].

Quicksort, discovered by (invented by?) C.A.R. Hoare, is an advanced method based on exchanging elements, as is Bubblesort, but its performance is the best general purpose method for sorting arrays known. It requires extra storage, but not overmuch.

The basic idea is that given an array we partition the elements about one of their number such that all emements to the 'left' of the chosen element are less than or equal to the chosen element, and all elements to the 'right' of the chosen element are greater than or equal to it. This partitioning process is then repeated on the two partitions, and on the partitions generated from that partition and so on. The recursive quality of this definition makes QUICKSORT programes written in a language which supports recursion very simple to understand, whereas in BASIC some extra housekeeping is needed which can tend to obscure the algorithm. The program in listing 5 uses an array to simulate a stack, upon which are placed the boundaries of partitions which have yet to be partitioned further (note that the partitioning stops when every partition consists of a single element). The actual array used is a 10x2, although we treat it simply as a 10 element array, each element containing two numbers, the left and right boundaries. SP stands for Stack Pointer, and points to the next partition boundaries to be partitioned (or equals 0, in which case the sort is finished). LR and RR stand for Left Request and Right Request respectively, being the boundaries of the current partition being operated on.

QUICKSORT's only real problem is with the worst case behaviour, which occurs when at each partition element selected to partition the array about is the largest value in the partition. In this case not only does the sorting become much slower but we can also have problems with the size of stack required, which at times may approach the size of the original array. This can be avoided using some extra coding to stack the larger partition and partition the smaller, in which case the length of stack needed can be limited to LOG$_2$(NE) (details in Wirth's book).

Timings

Each of th 5 methods was used to sort arrays with 2, 4, 8, 16, 32, 64, and 128 random numbers. The results are given below:

	Number of elements						
Met'd	2	4	8	16	32	64	128
Ins'n	.03	.12	.37	1.37	3.92	16.82	73.25
Ins'n ★	.03	.15	.50	1.50	6.23	19.52	82.00
Sel'n	.03	.13	.42	1.37	4.75	17.37	66.01
Exc'ge	.02	.17	.58	2.32	10.53	47.10	164.13
Q'sort	.13	.33	.83	1.85	4.3	9.95	21.48

★ Using the sentinel method.

Thus for anything over 64 elements QUICKSORT should be used, unless storage space is at a minimum, in which case the shortest of the other methods should be used, probably Bubblesort.

Sorting by Insertion (Listing 1).

```
10 POKE 59468,14
20 INPUT "NUMBER OF ELEMENTS TO SORT";NE
30 DIM FILE(NE)
40 FOR I=1 TO NE
50    FILE(I)=100*RND(1):PRINT FILE(I)
60 NEXT I
70 REM HERE COMES THE ACTUAL SORT.
80 FOR I=2 TO NE
90    X=FILE(I)
100    FOR J=I-1 TO 1 STEP -1
110       IF (X<FILE(J)) THEN FILE(J+1)=FILE(J):NEXT J:FILE(J+1)=X:GOTO 140
120    FILE(J+1)=X
```

```
 130     J=1:NEXT J
 140 NEXT I
 150 PRINT:PRINT
 160 FOR I=1 TO NE
 170 PRINT FILE(I)
 180 NEXT I
READY.
```

Insertion Sort with Sentinel (Listing 2)

```
 10 POKE 59468,14
 20 INPUT "NUMBER OF ELEMENTS TO SORT";NE
 30 DIM FILE(NE)
 40 FOR I=1 TO NE
 50     FILE(I)=100*RND(1):PRINT FILE(I)
 60 NEXT I
 70 REM HERE COMES THE ACTUAL SORT.
 80 FOR I=2 TO NE
 90     X=FILE(I):FILE(0)=X:J=I-1
 100     IF (X<FILE(J)) THEN FILE(J+1)=FILE(J):J=J-1:GOTO 100
 110     FILE(J+1)=X
 140 NEXT I
 150 PRINT:PRINT
 160 FOR I=1 TO NE
 170 PRINT FILE(I)
 180 NEXT I
READY.
```

Sorting by Selection (Listing 3).

```
 10 POKE 59468,14
 20 INPUT "NUMBER OF ELEMENTS TO SORT";NE
 30 DIM FILE(NE)
 40 FOR I=1 TO NE
 50     FILE(I)=100*RND(1):PRINT FILE(I)
 60 NEXT I
 70 REM HERE COMES THE ACTUAL SORT.
 80 FOR I=1 TO NE-1
 90     K=I:X=FILE(I)
 100     FOR J=I+1 TO NE
 110         IF (FILE(J)<X) THEN K=J:X=FILE(J)
 120     NEXT J
 130     FILE(K)=FILE(I):FILE(I)=X
 140 NEXT I
 150 PRINT:PRINT
 160 FOR I=1 TO NE
 170 PRINT FILE(I)
 180 NEXT I
READY.
```

Sorting by Exchange (Listing 4).

```
 10 POKE 59468,14
 20 INPUT "NUMBER OF ELEMENTS TO SORT";NE
 30 DIM FILE(NE)
 40 FOR I=1 TO NE
 50     FILE(I)=100*RND(1):PRINT FILE(I)
 60 NEXT I
 70 REM HERE COMES THE ACTUAL SORT.
 80 FOR I=2 TO NE
 90     FOR J=NE TO I STEP -1
 100         IF (FILE(J-1)>FILE(J)) THEN X=FILE(J-1):FILE(J-1)=FILE(J):FILE(J)=X
```

```
 110    NEXT J
 140 NEXT I
 150 PRINT:PRINT
 160 FOR I=1 TO NE
 170 PRINT FILE(I)
 180 NEXT I
READY.
```

Quicksort (Listing 5).

```
 10 POKE 59468,14
 20 INPUT "[illegible]UMBER OF ELEMENTS TO SORT[illegible]";NE
 30 DIM FILE(NE),STACK(10,2):LEFT=1:RIGHT=2:REM CONSTANTS.
 40 FOR I=1 TO NE
 50    FILE(I)=100*RND(1):PRINT FILE(I)
 60 NEXT I
 70 REM HERE COMES THE ACTUAL SORT.
 80 SP=1:STACK(SP,LEFT)=1:STACK(SP,RIGHT)=NE
 85 LR=STACK(SP,LEFT):RR=STACK(SP,RIGHT):SP=SP-1
 90  I=LR:J=RR:X=FILE(INT((LR+RR)/2))
 100   IF (FILE(I)<X) THEN I=I+1:GOTO 100
 110   IF (X<FILE(J)) THEN J=J-1:GOTO 110
 120   IF (I>J) THEN 130
 125    W=FILE(I):FILE(I)=FILE(J):FILE(J)=W:I=I+1:J=J-1
 130  IF (I<=J) THEN 100
 135 IF (I<RR) THEN SP=SP+1:STACK(SP,LEFT)=I:STACK(SP,RIGHT)=RR
 140 RR=J
 145 IF (LR<RR) THEN 90
 150 IF (SP<>0) THEN 85
 155 PRINT:PRINT
 160 FOR I=1 TO NE
 170 PRINT FILE(I)
 180 NEXT I
READY.
```

Relocation.

Relocation is the process of moving a program or part of a program (in this article always a machine code program) from its original location to a new one, such that it still works and delivers the same results as the original. We may have a program written for an 8K machine and wish to relocate it for a 32K machine, or we may want to produce a gap in the middle of a program to accommodate a new section of coding by moving part of the original program up or down.

In this article and the next issue's PET Corner a program to accomplish this will be given, together with the method of linking it into the Commodore Machine Code Monitor.

Consider the small section of programming (which accomplishes nothing at all) below:

```
0380 A9 05      LDA     $05       ;*
0382 A8         TAY               ;*
0383 BE 91 03   LDX     $0391,Y   ;
0386 96 80      STX     $80,Y     ;*
0388 88         DEY               ;*
0389 10 F8      BPL     -$08      ;*
038B 8D 50 40   STA     $4050     ;*
038E 4C 80 03   JMP     $0380     ;
0391 8A
0392 F1
0393 23
0394 69
0395 14
0396 00
```

and imagine that we want to relocate the program so that is starts a location $0350 instead of $0380, and that we will refill the space vacated with zeros. If we write a short BASIC statement to move the program itself:

```
FOR I=3*256+8*16 TO 3*256+9*16+6:POKE I-3*16,PEEK(I):NEXT I
```

and then fill the vacated space with zeros will the relocated program still work (i.e., perform the same task as it did before)? Those instructions marked with an asterisk will still work correctly in their new location since they either do not reference a memory location (e.g. TAY, DEY), reference a memory location outside the range of the program relocation (e.g. STX $B1,Y or STA $4050) or reference a memory location not in an absolute manner but in a relative one (e.g. BPL -$ 08) where the memory location referenced is not fixed but is rather the one 8 bytes before the instruction after the BPL instructions.

The two instructions at $0383 and $038E must however be changed in order for the program to work correctly. In this case, since the program has been moved down in the memory by #$30 bytes, this results in the instructions becoming:

0353 BE 61 03 LDX $0361,Y
and
035E 4C 50 03 JMP $0350

Had we moved the program up by #$20 bytes the instructions would have become:

03A3 BE B1 03 LDX $03B1,Y
and
03AE 4C A0 03 JMP $03A0

i.e. we subtract (if the program is to be moved down) or add (if it is to be moved up) the displacement of the program from any address referencing locations within the program. Since we will concern ourselves only with programs written for locations in the second and subsequent pages of RAM (or ROM) only instructions which are 3 bytes long need be considered for alteration, and will be altered only if their second and third bytes (low/high) form an address within the program to be relocated.

In this issue a COPY routine will be listed, which will take three addresses from a monitor command of the following form:
C (oldfrom), (oldto), (newfrom)
where each address within brackets is a 4 digit hexadecimal address, e.g.
C 0380, 0396, 0350
will move the program from locations $0380-$0396 to a new location starting at $0350. COPY will be written as a subroutine which will be called by the relocator program to move the program without any relocation taking place, then the actual relocation will be performed on the new program.

The main reason for splitting the task up like this is that we may on occasions want to move data (or even a program) without altering any of the bytes moved (this is most important in the case of data, where a data byte might be interpreted as an LDA instruction and the two bytes following it altered). Having two functions C, which doesn't alter anything moved, and T (for Transfer), which does, makes things a lot easier. A listing for Copy follows:

```
mover routine

B*
    PC  IRQ  SR AC XR YR SP
.  0401 E62E 32 04 5E 00 F6
.
.  033A 18 00 48 20 EB E7 20 A7
.  0342 E7 68 90 09 AA A5 FB 95
.  034A B1 A5 FC 95 B2 60 A9 00
.  0352 20 3C 03 90 E3 A9 02 20
.  035A 3C 03 90 F7 A9 04 20 3C
.  0362 03 90 F7 38 A5 B1 E5 B3
.  036A A5 B2 E5 B4 B0 CA EA EA
.  0372 EA 38 A5 B3 E5 B1 85 B7
.  037A A5 B4 E5 B2 85 B8 A5 B1
.  0382 85 BB A5 B2 85 BC A2 00
.  038A 86 BA E8 86 B9 A5 B6 C5
.  0392 B2 30 23 D0 06 A5 B5 C5
.  039A B1 30 1B A9 FF 85 B9 85
.  03A2 BA A5 B3 85 B1 A5 B4 85
.  03AA B2 18 A5 B5 65 B7 85 B5
.  03B2 A5 B6 65 B8 85 B6 E6 B7
.  03BA D0 02 E6 B8 A0 00 B1 B1
.  03C2 91 B5 18 A5 B1 65 B9 85
.  03CA B1 A5 B2 65 BA 85 B2 18
.  03D2 A5 B5 65 B9 85 B5 A5 B6
.  03DA 65 BA 85 B6 A5 B7 D0 02
.  03E2 C6 B8 C6 B7 A5 B7 05 B8
.  03EA D0 D4 38 00 FF FF 00 3F
```

If space is available in the next issue a complete disassembly of the routine will be given, but until then the hexadecimal dump above should suffice. The routine may be entered into a second cassette buffer. taking locations $033A to $03ED. The listing presented uses BRK instructions (hexadecimal $00) at $033B and $03ED to return control to the monitor. In the final program these will be replaced by RTS instructions ($60). To pass back information about the usccess or otherwise of the routine in doing its appointed task the carry bit is used. If return is made through $033A then the carry bit is cleared, indicating to the calling routine the some error, eg. (oldto) less than (oldfrom), was encountered. The return through $03EC sets the carry, indicating success.

To use the routine at present load the routine from the monitor and enter the command:
G 0350, (oldfrom), (oldto), (newfrom)
and the memory locations from (oldfrom) to (oldto) inclusive will be moved so as to start at (newfrom).

Details will be given in the next issue to convert the routine into a form suitable for old ROMs.

Machine Code Hints.

1. To detect whether 2 or more bytes are all zero **or** the memory bytes together. If the zero flag is set when the last byte has been or'd in then all the bytes were zero, e.g.

```
03E6 A5 B7      LDA $B7
03E8 05 B8      ORA $B8
```

sets the zero flag if $B7 and $B8 were both zero.

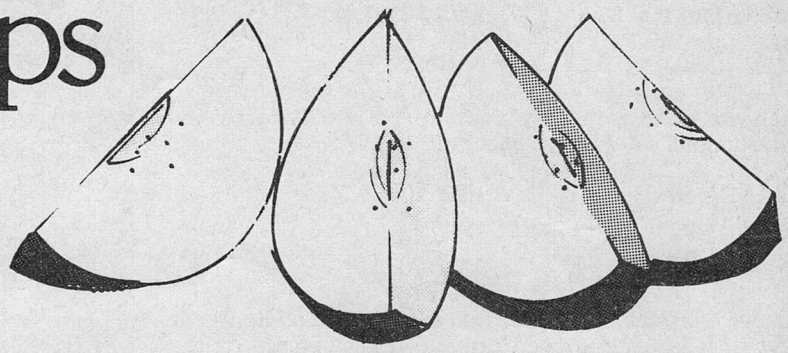
2. To decrement a 2 byte integer by 1, check the low order byte for zero. If it is decrement both the high order and the low order bytes, else only decrement the low order byte. Thus if $B7, $B8 (low/high) contain a 2 byte integer to be decremented the following must be used:

```
03DEA5   B7   LDA $B7   ;Check low order byte.
03E0 D0  02   BNE +$02 ;Branch if it is not zero.
03E2 C6  B8.  DEC $B8  ;Decrement high order byte
03E4 C6  B7   DEC $B7  ;Decrement low order bvte.
```

Putting the two short routines together (and their addresses are contiguous) yields a short routine which decrements $B7, $B8 and returns with the zero flag set if the result was zero, a useful little routine for creating long loops in machine code.

Using the page-zero, X addressing mode means that a number of counters can be set up in page-zero and a routine written to decrement any one of them by calling it with the relevant value in the X register.

Apple Pips



Compiled by

Roy Stringer,
Sales Manager,
Microdigital

NEW PRODUCTS

Apple Computers have produced a couple of extremely useful programmes recently which exemplify the high quality that we have come to expect from all Apple products.

Visicalc

This is an excellent general purpose programme which moulds itself to your own calculating requirements by simulating a calculator, a pen and a sheet of paper. The programme uses a mapped notesheet on which you can 'jot' figures, totals, tables of variables etc., and literally play about with them. For instance, you can produce very complex equations using many variables and, by changing any one or any group of them, see instantly what happens to the rest.

Variables may be functions of variables at other locations on the page, so that if you are using the page to display a table of retail prices of items, then each price would be entered as a function:- See Table 1 below.

The function in Columns D and E of Table 1 may be as complex as you like, but are only displayed as a value of the page. By placing the cursor on a location, however, you can read the function which produced that value at the top of the screen.

If the VAT rate in location B1 were changed, then every figure on the page using that location, e.g. E6, would be instantly modified.

Altogether, this is a pretty useful programme.

Apple Writer

This is an extremely well written Text Editor programme which is so cheap that I can almost see its being put onto the Disc II Master Discettes, by the end of the year.

Here is a list of some of its main features:

1. Does not require a lower case adaptor, but instead converts the outgoing ASC II Code into the correct code for the printer.

Table 1.

	A	B	C	D	E	F	G
1	VAT RATE	0.15		MARK UP 1	0.16	MARK UP 2	0.21
2		ITEM	COST	SELL	INC VAT		
3		COPPER PAN	25.02	C3+(C3*E1)	D3+(D3*B1)		
4		TIN PAN	15.95	C4+(C4*E1)	D4+(D4xB1)		
5		FISH SLICE	3.25	C5+(C5xE1)	D5+(D5xB1)		
6		CUP HOOK	0.13	C6+(C6xG1)	D6+(D6xB1)		
7		KNIFE	0.95	C7+(C7xG1)	D7+(D7xB1)		
8		FORK	0.85	C8+(C8xG1)	D8+(D8xB1)		

2. A reverse Display Character produces upper case on the printer. A Normal Display Character produces lower case on the printer.

3. Up to twelve screens full of text can be manipulated in memory at once.

4. Text scrolls both upwards and downwards.

5. Easy inserting and deleting of text.

6. Full control of printer formating from within the text itself. Printer commands are not printed.

7. Print Constants "set up" programme which allows you to set up the following formating variables:

   a. Left margin
   b. Right margin
   c. Top margin
   d. Bottom margin
   e. Number of lines per page
   f. Page Headings
   g. Page numbers
   h. Single page or Continuous Printing, to allow insertion of separate sheets of paper
   i. Number of spaces between lines
   j. Set or Not Set Bit 8 - this makes the Apple Writer software compatible with a large range of printers.

Apart from these constants, other formating modes can be used from within the text as described above, namely Left Justify, Right Justify, Centre Justify and Fill Justify.

The Apple Writer also has a search routine which will place the cursor in front of every occurence in turn, of a pre-specified group of characters. This routine can also be used to replace each occurance with another set of characters either automatically or on individual request as it finds each occurence.

The Appple Writer and Visicalc are quite definitely high quality programmes, of which I have only been able to provide a thumbnail sketch here. Perhpas we will see a major article on these in a future issue of this magazine

DISC 11

The latest supplies of Apple 'Disc 11' Drives are arriving with a disc pack containing three discs:

System Master Dos 3.2 Standard
System Master Dos 3.2 Plus
1 Blank Discette

The "standard" master has a selection of programmes written in Integer Basic while "Plus" master has a selection of Applesoft Programmes.

Unfortunately, no where on these discs can one find a disc version of Integer Basic which makes the standard disk quite useless unless an integer card is purchased. However, I think that most dealers should be able to provide their customers with a copy if you ask them nicely. (Let's have it back please Apple!!!)

Another utility which Apple have not put on their masters yet, but which ITT have been supplying for some time, is a slot to slot copy programme which allows copies to be made from any drive connected to the machine to any other connected drive. This utility is essential for people who purchase systems to implement software which requires, or is more efficient with, two Drives on Different slots.

Apart from this, a slot to slot copy is about three times faster than a Drive to Drive Copy.

Again if your Apple dealer is also an ITT dealer, he should be able to let you have a copy of this programme.

This programme is a demo for switching between HGR and HGR 2 as used by APPLEVISION. It can be extremely useful for moving-graphics programmes.

```
5  REM  SCREEN-SWOP DEMO
10  REM BY N.KELLY  5/4/80
15  REM 'CALL 62450' IS APPLESOFT
20  HGR : HCOLOR= 7
30  REM START ON SCREEN'1'
40  POKE  - 16302,0
45  REM NOW FULL SCREEN
50  REM GOTO 100
100  GOSUB 1000: REM DRAW A '1'

110  POKE  - 16300,0
115  REM SWITCH SEEN SCREEN TO'1'
120  POKE 230,64
125  REM NOW THINKS IS ON 'HGR2'
130  CALL 62450
135  REM CLEARS 'CURRENT' SCREEN
140  GOSUB 2000: REM DRAW A '2'
150  POKE  - 16299,0
155  REM SWITCH SEEN SCREEN TO'2'
160  POKE 230,32
165  REM NOW THINKS IS ON 'HGR1'
170  CALL 62450
175  REM CLEARS 'CURRENT' SCREEN

180  GOTO 100
999  END
1000  HPLOT 100,15 TO 100,90: RETURN

1001  REM DRAW A '1'
2000  HPLOT 75,10 TO 125,10 TO 75,
      100 TO 125,100: RETURN
2002  REM DRAW A '2'
```

This programme tells you how many free sectors are still available on a disc.

It is useful for checking if there is enough room to put a large programme onto it, or for telling you when to buy your next box.

```
5  REM DISC FREE SPACE,48K,DOS3.2
```

```
6  REM ERR CORRECTED NK 17/2/80
10  FOR I = 768 TO 801: READ B: POKE
    I,B: NEXT
20  DATA 32,247,175,169,0,133,6,1
    33,7,160,56
30  DATA 185,187,179,162,7,10,144
    ,6,230,6,208,2
40  DATA 230,7,202,16,244,200,192
    ,196,144,234
45  DATA 96,63
60  CALL 768
70 FR =  PEEK (6) + 256 *  PEEK (
    7)
80  PRINT : PRINT FR;"  SECTORS F
    REE": END
```

This programme prints every control character that it finds as a FLASH character.

```
1   REM    PICKLE
2   REM    FOR FINDING FORGOTTEN
3   REM     CTRL CHR'S IN PROGRAM
        NAMES
10   DATA 201,141,240,21,201,136
20   DATA 240,17,201,128,144,13
30   DATA 201,160,176,9,72,132
40   DATA 53,56,233,64,76,249
50   DATA 253,76,240,253
60   FOR I = 768 TO 768 + 27
70   READ V: POKE I,V: NEXT I
80   POKE 54,0: POKE 55,3
90   CALL 1002
100   END
110   REM  TO REINSTATE NORMAL
115   REM  TYPE'PR#0'
```

N.B. After it has been run, the message 'SYNTAX ERROR' becomes 'SYNTAX ERRORG' with a flashing 'G' instead of a "BEEP".

APPLE VERSUS ITT 2020

many people are not aware of the actual simularities and differences between these two machines and indeed find it hard to believe that ITT make Apples under licence and not vice versa. Here is a list of the main differences between the two:-

1. APPLE is supplied with V & W Video output. ITT is supplied with P.A.L. colour modulated output. A B & W modulator is available for the Apple at £14.00 and a digital colour card at £79.00, although the latter are still very difficult to obtain.

2. The APPLE is supplied with the Autostart F8 ROM and the ITT is still supplied with the old F8 ROM. Autostart ROMs cost £38.00.

3. HGR on the ITT is 315 x 192 addressable dots, APPLE is 280 x 192 dots. This causes one blank vertical line in every nine, when using Apple Hi Resolution graphics programmes on the ITT.

4. The ITT does not understand the Applesoft Basic command:-

HPLOT TO x,x

but requires

HPLOT y,y TO x,x

5. The ITT is not supplied with paddles, although they may be purchased as an extra for about £16.00

Thanks to Nigel Parry of Microsense for the following useful information.

PROGRAM RECOVERY

I hope the following small program will be of some assistance someday to those of us with clumsy fingers.

If you accidentally type 'NEW' or 'FP' then this could save a few blisters and possibly a late night by recovering your deleted program in the state it was before you destroyed it. It is not guaranteed to work perfectly, but any attempt that succeeds will be better than nothing!!!

It is important to note that the entire program must be typed in IMMEDIATE EXECUTION MODE (with no line numbers), otherwise your lost program will never be recovered. Further more, upon completion of the recovery, the program should immediately be saved, as RUN, or any command which alters the program (ie inserting a line) will destroy your system and require to be re-booted.

TO RECOVER A NEW'D OR FP'D PROGRAM

LOMEM:38200 (assuming 48K system)

```
X = PEEK(103) + PEEK(104) * 256  : Z = X + 5

FOR A = Z TO Z +  275 :  IF PEEK(A)<> 0 THEN NEXT

A = A + 1 : HI = INT( A / 256 ) : LO = A - HI * 256

POKE X , LO  :  POKE X + 1 , HI
```

At this point type LIST to see if your program has re-appeared. (I would let it list from top to bottom in case CTRL C destroys it !!), if it has then SAVE it, if not then try again and again.

As I said, I don't guarantee the results, so just try it and see!!!

RELOCATED INTEGER.

A relocated version of RAM Based INTEGER is now available, which is based on the standard Integer Basic, but amended to reside at the top of available RAM memory.

This avoids the common problem associated with the use of RAM Integer Basic to run software written on and for the original ROM Integer machines; whereby the interpreter is overwritten by Page 1 of high resolution graphics, and/or it is overwritten by machine language sub routines which are frequently stored at location $1000, the starting address for standard RAM Integer Basic.

Using RELOCATED INTEGER, any programme written for the Apple II, should have a better chance of running on an Apple II Plus/Europlus or ITT Palsoft machine, within the constraints, of course, of available memory size.

RELOCATED INTEGER differs from the standard RAM Integer by remaining co-resident in memory, giving the facility to switch between Basic languages at will, and in addition the Mini-Assembler facility (sacrificed from Floating-Point machines because of ROM size restraints) is provided, also co-resident in memory.

On DISC SYSTEMS Relocated Integer makes use of the DOS 'INT' and 'FP' commands to switch between Basics, thus giving automatic language change under programme request, in the same way as if an Integer Firmware Card were fitted.

On CASSETTE SYSTEMS Relocated Integer provides the following command structure:

-Enter Integer Basic (RELOCATED INTEGER)-cleared for new programme.
-Enter Floating-Point (Applesoft/Palsoft) Basic-cleared for new programme.
-Enter Monitor.
-Enter Assembler (mini-Assembler).
-Resume Basic (with programme intact) from Monitor/Assembler/Edit.
-Edit Mode clears screen and sets text window to width 33 (which forces both Basics to abandon the indented format for listing) facilitating the editing of Basic programmes.

MERSEYSIDE MICROCOMPUTER GROUP

APPLE SPECIAL INTEREST GROUP

I have a disk which contains the following programs:-

MORTGAGE - FROM PCW SERIES

BIGTILL - FROM PCW SERIES

FORMATTER - FROM FINDLEY AND WATT

PL/0 - FROM ALGORITHMS + DATA STRUCTURES = PROGRAMS

CHEQUEBOOK - A FORTHCOMING ARTICLE IN COMPUTER AGE

A NUMBER OF SMALL DEMONSTRATION GRAPHICS PROGRAMS

and am prepared to make them available on an "as is" basis. Since no program exchange scheme can work without input from all its potential users please send a disk which contains the source of at least one useful program for inclusion on the disk, and I shall return the disk to you with the latest collection of contributed programs copied on to it. If there is sufficient demand I shall consider providing the same service for Basic and Assembler programs.

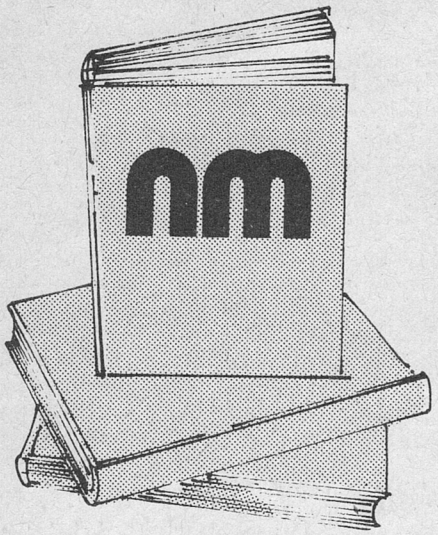
Dr Martin Beer
Computer Laboratory
University of Liverpool
P O Box 147
Liverpool
L69 3BX
Tel: 051-709-6022 Ext 2967

UNIVERSITY OF LIVERPOOL.
MICROPROCESSOR LABORATORY.
MICROCOMPUTER ASSEMBLER PROGRAMMING.
5 AFTERNOONS, STARTING TUESDAY 20TH MAY 1980.

This course gives a practical introduction to assembler level programming of microcomputer systems, particularly those based on the extremely popular 6502 processor. Practicals are arranged around the AIM-65 single-board computer, but the principles discussed can be applied with equal ease to the APPLE and PET, which are also based on the 6502.

For prices and further information, Please contact:

DR. M. D. Beer,
Computer Laboratory,
University of Liverpool,
P.O. Box 147,
LIVERPOOL.
L69 3BX.



NASCOM NOTES

COMPILED BY M.SHANAHAN
ENGINEERING DEPT. MICRODIGITAL LTD

THIS month most of the notes come from our readers.

The first letter I have received comes from Mr B Lounds of Heysham High School in Morecambe, and concerns the problem cited in the January Gazette using 8 K BASIC and certain printers.

The BASIC Interpreter keeps the line length in location 4162. This is normally 47, but can be changed using the 'WIDTH' command. The Interpreter also keeps a running total of the characters per line in location 4267. It increments the count for each character typed, and when this equals the line length in location 4162 the Interpreter gives out a Carrage Return and Line Feed.

The problem was that the printer required an 'ESC' character when changing its printing format, but 8K BASIC counts the Escapes as real characters (it also counts Backspaces).

This means that after 48 'characters' the Nascom does a Line Feed, despite there being less than 48 visable characters on the line!

An easy enough cure for this (easy enough when you know what's causing it!) it to reset the Character Counter after outputing every 'ESC'. Try the program below:-

```
10 for I=1 to 1000
20 Print I;
30 Print CHR$(27);
35 POKE 4267,0
40 NEXT I
```

Line 35 resets the counter after printing the ASCII code for Escape.

To get round the Backspace problem simply subtract one from the Character Counter, see below:-

```
10 FOR J=1 TO 47
20 PRINT "X";
30 FOR Q=1 TO 300:NEXT Q
40 PRINT CHR$(8)
50 V=PEEK(4267):POKE 4267, V-1
60 PRINT" ";
70 NEXT J
```

Try it with and without line 50!

My thanks to Mr Lounds and to Mr D A Boyd who also sent in the cure.

NEXT we have a short review, by John Haigh, of Nascom's Text Editor NASPEN. John is an avid keyboard clouter and has been known to write articles for this very Journal (see L.S.G. issues 1 and 2!!!) He also is the author of JETSET in this month's mag.

NASPEN

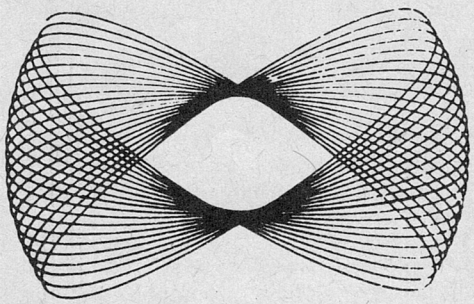
Naspen is Nascom's text editing and formating system, supplied in two 2708 EPROM's to reside between $B800 and $BFFF. The program provides over forty commands to control the entry, insertion and deletion of text, editing, formating and printing; it also reverses the operation of the shift key, so that it operates as on a normal typewriter, and provides automatic key-repeat. The system can produce very tidy text, left- and right-justified to any required line width, which gives the output a very professional appearance. The documentation supplied with the program is rather uninspired, but it does give a reasonably complete description of each command.

The main problem I have encountered while using Naspen is the absence of any paragraph formating command. If you enter a paragraph in the usual way, by typing a 'newline' at the end of a sentence and indenting the following line, when the text is formated the 'newline' is removed prior to the adjustment of line lengths, and the usual result is a blank space in the middle of a line instead of a paragraph. The easiest way around this is to put two 'newlines' in front of a paragraph; such newline pairs are not removed by the formating, but the result is that paragraphs are separated by a blank line. The effect is not unpleasant - it can be seen in the INMC newletter which is produced with the aid of Naspen - but it is not standard practice. To produce normally spaced paragraphs you have to use a single 'newline', bracketted by formating delimiters, which means that you have to press a total of nine keys just to generate a new paragraph.

However, the ease with which typing errors can be edited out of the text and the neat text that can be produced even by unskilled typists outweigh such minor irritations. Naspen is particularly valuable for repetitive typing tasks, as text can be stored on tape and could be built up from standard sections and then edited to give a 'customised' product.

Finally, a note on last months RAM tests for Nascom 2 owners. As Graham Myers point out elsewhere in this issue, I neglected to say that 2 byte calls could be used to the NAS-SYS routines. The advantage of using these calls is that they are monitor independent. This means that if someone out there decides to write NAS-SYS MK 2 (God help us) the program will still run, without the modifications I had to do to change form T4 to NAS-SYS. There is a disadvatage though. The NAS-SYS 'Restart Calls' are not real instructions in the true sense. They are really one byte restart instructions followed by one byte of data, which is used by the monitor routine to find out which location you were trying to call. This is great for Nascom owners, but if a 'non Nascom Z80 freak' is trying to follow an uncommented assembly, he could have problems. The same goes for the other Restart instructions used by NAS-SYS (or, for that matter, the 'Print String' subroutine used in T2 and T4). I'm not saying don't use them in your programs, but do document the listing sufficiently. I hope this is controversial enough to stir some of you to write in and disagree with me. A last note on the RAM test. It shouldn't be needed in the future, because Nascom have informed us that any new deliveries of Nascom 2s will be without the ill-fated 16K RAM board, but will have 4118's on board. A new design 48K RAM board should be available soon, hopefully without free helpings of Memory Plague!

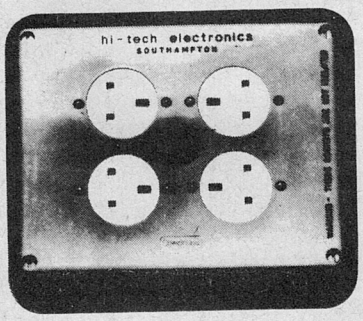
Thats all for this month.



hi-tech electronics

1 Richmond Gardens, Highfield Southampton SO2 1RY
Telephone (0703)555072

PROGRAMMABLE MAINS BOX



* Moulded Plastic Box contains four 13A sockets and circuitry for computer controlled mains switching and dimming applications.
* Three on/off sockets and one 32-level dimmer socket.
* Uses the latched parallel output port from your computer.
* TTL compatible.
* Each socket controls up to 6A - total load 13A.
* All sockets normally live, and so may be used as an ordinary distribution box when not connected to your computer.
* Each socket provided with individual mains suppression for minimal interference.
* Fully isolated opto-coupled computer interface.
* Fully built and tested.
* Full documentation supplied.
* Price, including P & P: £69.00 plus VAT.

Nascoms Programs & Information

a book by

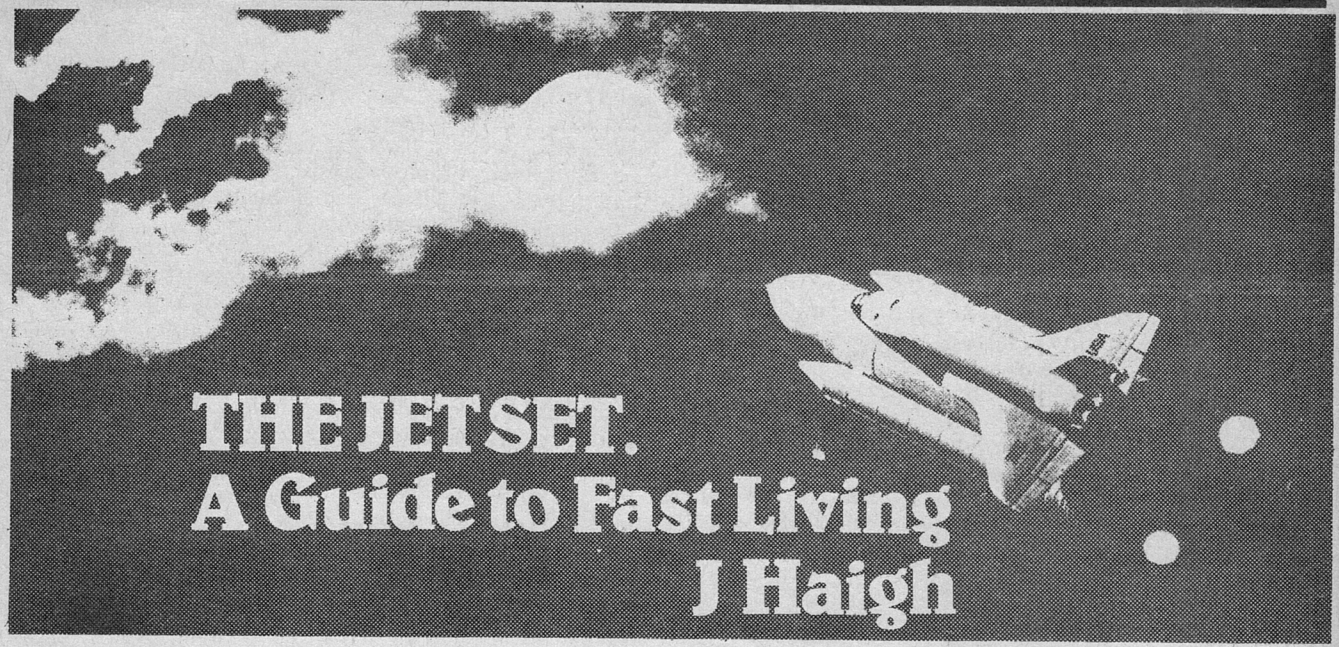
Merseyside Nascom Users Group

ONLY **£2.25**

Including P & P

Yours at ...

Microdigital Limited
25 Brunswick Street
Liverpool
L2 0PJ



THE JET SET. A Guide to Fast Living

J Haigh

LIFE is a computer simulation which follows the evolution of cell configurations under the control of a simple set of rules. This phrase should be memorised and repeated to anyone who catches you idly watching the fascinating patterns which a life program produces on the VDU screen. The rules for the birth, death and survival of cells can be varied at will, but the richest range of behaviour seems to be obtained with the 'genetic laws' of J.H. Conway. In conway's universe the cells are arranged in a square array, and the future of any square is determined by the number of live cells in the eight squares which border it. The rules are very simple - a live cell will only survive in the next generation if it has two or three neighbours alive, and a cell is born in any empty square with three live neighbours.

The variety of patterns produced by such simple rules is amazing. Some configurations die out rapidly, others reach stable shapes, oscillate, move or grow to an infinate size. Changing the position of one cell pattern will completely transform the course of it's evolution. For example, of the twelve possible rookwise -connected five cell patterns six quickly die out, two become stable, three turn into the period-two oscillator known as traffic lights, while the notorious R-pentomino (figure 1) takes over eleven hundred generations to produce a central region with a population of 86, which oscillates but does not move, plus six gliders. A glider (figure 2) is a five-cell pattern which moves diagonally across the array.

Several machine code programs have been published recently, but they all work on small arrays (typically 20x30) and are slow, scanning roughly 2000 cell-generations per second on a system running at 2 Mhz. The program described below runs on a 125x125 array and is at least ten times as fast. The whole of the evolution of the R-pentomino can be followed up to the point where the population stabilises, taking approximately four minutes at 4 Mhz.

The pattern is stored in a block memory, two cells to a byte. The most significant bit of each nibble is 1 if the cell is alive. The bottom three bits are used to count the number of live neighbours of the cell. The array is scanned, each empty byte being ignored. When a non-zero byte is found the program checks of either or both cells are alive. For every live cell one is added to each of it's eight neighbours. If a live cell mapped into the low nibble of a byte has eight live neighbours the final count would cause overflow into the high nibble; this is avoided by reseting bit 3 (line 1930). When the scan is completed the cells at the edge of the array are inspected for overflow. The overflow routine first tests for gliders leaving the array, which are counted and erased; any other type of overflow is flagged of O/FL printed on the top line of the screen. Once overflow occurs edge effects distort the course of the evolution; gliders just keep moving unless they run into something, so gliders that leave the array will not usually affect the rest of the population (unless two gliders produced by the pattern could collide outside the array).

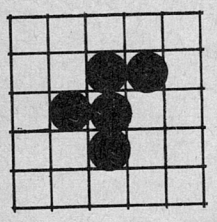
A second scan is now carried out to convert the results from the first scan into the next generation, following the rules for evolution encoded in subroutine EVOLVE. By changing this subroutine you can try other genetic laws - for example, 3-4 life, in which a cell must have three or four neighbours for birth or survival. After the second scan the new generation is displayed and the process is repeated. If a pattern dies out of becomes stable, or if the space bar is held down at any time, the program stops and waits for instructions from the keyboard. Key R restarts the program with a clear screen, M restarts but leaves the current pattern on the screen so that it can be modified. The four cursor control keys on the Nascom 2 move the whole display relative to the array, so that although the display is only 45x96 you can look at any region in the full area; the program can be run with the display in any position. Evolution is started by pressing the space bar; by pressing and holding down the space bar you can single-step through a life history.

The program listing is for a Nascom 2 with a optional graphics ROM. If you haven't got the graphics ROM, or if you have a Nascom 1, the following changes should enable you to run the program on your system. Firstly, for the Nascom 1 the keyboard and character input routines are handled differently. Lines 490 and 2880

should be changed to CALL £0069 (CD 69 00), and line 2910 to CALL £003E (CD 3E 00). Secondly, the Nascom 1 does not have cursor control keys; any four keys can be used if their ASCII values are put in the key function table, lines 4850 to 4890. The third set of changes relate to the display. A graphics board is available for the Nascom 1 from Bits and P.C.s of Wetherby; the extra characters are held in an EPROM and a version can be obtained with a six-pixel graphics set similar to that of Nascom 2. Patches suitable for such a system, with the pixel-graphics at £AO-£BF and £EO-£FF, are given after the main listing. If you have no graphics capability you can obtain a 30x48 display by using characters £27, £2C and £7C to display two pattern lines per screen line. The necessary changes are outlined below. To adapt the program to any other graphics system the sections to be changed are WRITE (lines 540 to 800), COPY (lines 1240 to 1590) amd DISPLA (lines 3560 to 3980).

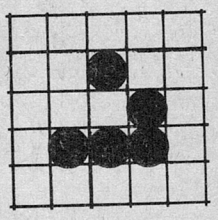
The program starts at £OC80 and needs at least 8K of extra memory, most of which is used for the workspace. If you have more memory you can add a few frills to the program. Useful extras are a routine to print out the screen display, which can be controlled by a command added to the section KBD at line 2910, and a method of adding a series of common patters, such as the glider, block, gun, etc., at any point on the screen, controlled by further keys added to the table at line 4790.

Some interesting patterns to follow are shown in the accompanying figures. When you have tried these you will have a good idea of the variety of changes which are possible, and you should have no difficulty in producing patterns of your own.

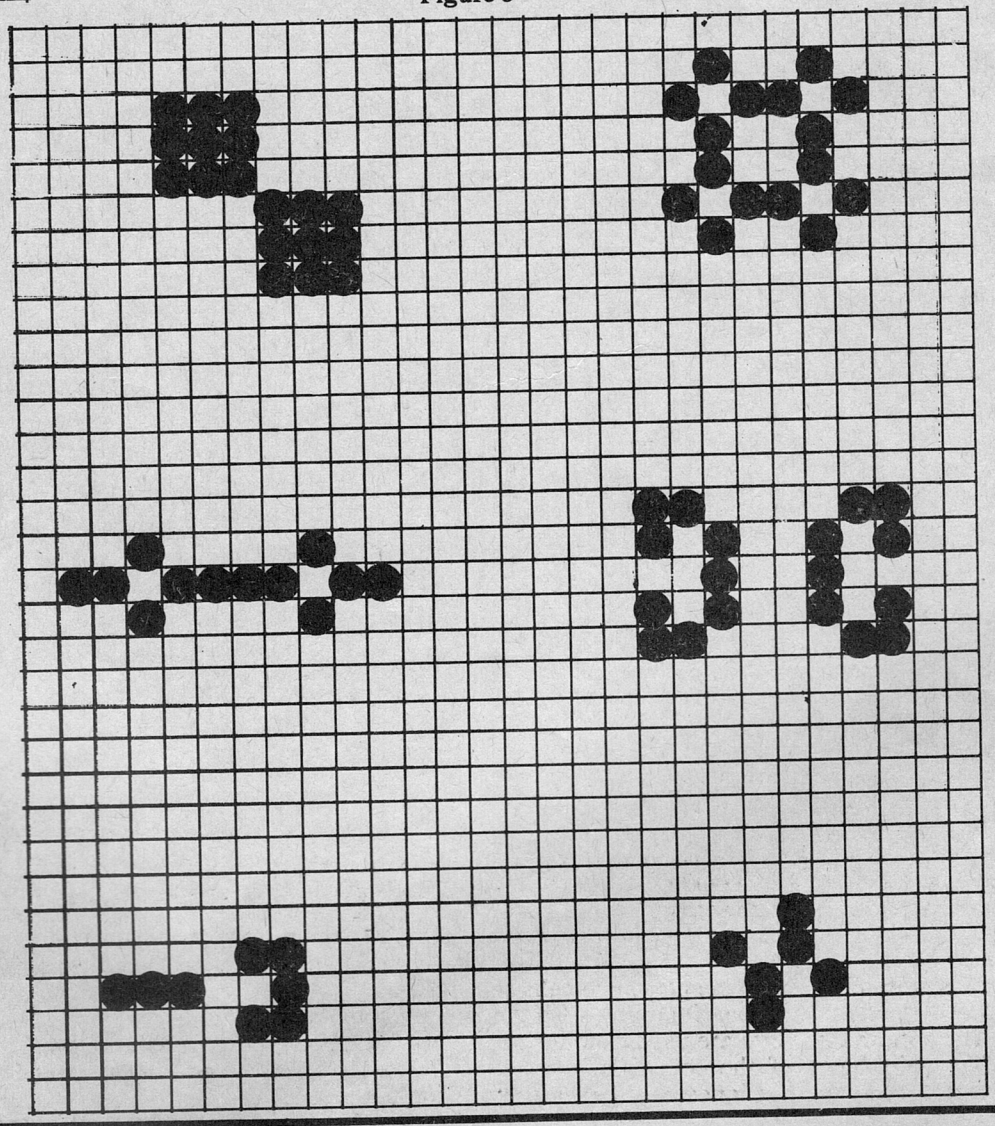


The R Pentomino
Figure 1

The Glider
Figure 2



Some simple oscillators
Figure 3



A Predecessor to the Glider Gun
Figure 4

```
OCE5 23          0590          INC  HL          ; NEXT ENTRY
OCE6 23          0600          INC  HL
OCE7 BE          0610          CP   (HL)       ; ENTRY FOUND?
OCE8 20FA        0620          JR   NZ WRT1    ; IF NOT, CONTINUE
OCEA EB          0630          EX   DE HL      ; CURSOR ADDRESS TO H
OCEB 13          0640          INC  DE         ; INCREMENT POINTER
OCEC 1A          0650          LD   A (DE)     ; TO FIRST BYTE
OCED A7          0660          AND  A          ; IS IT ZERO?
OCEE 2018        0670          JR   NZ COMAND  ; IF NOT, A COMMAND
OCF0 13          0680          INC  DE         ; GO TO NEXT BYTE
OCF1 1A          0690          LD   A (DE)     ; THIS SECTION OF CODE
OCF2 CB57        0700          BIT  2,A        ; UNRAVELS NASCOM 2'S
OCF4 280E        0710          JR   Z XORA     ; GRAPHICS
OCF6 57          0720          LD   D A        ; SAVE ENTRY IN D
OCF7 7E          0730          LD   A (HL)     ; LOOK AT BITS 2 AND 3
OCF8 E60C        0740          ANL  0C         ; OF CHARACTER AT (HL)
OCFA EE0C        0750          XOR  0C
OCFC 2005        0760          JR   Z XORD
OCFE AA          0770          XOR  L
OCFF 2802        0780          JR   Z XORL
OD01 1E20        0790          LD   A 20       ; BIT 5 IS INVERTED IF
OD03 AA          0800          XOR  D          ; BIT 2 IS 1, BIT 3 IS 0
OD04 AE          0810          XOR  (HL)
OD05 77          0820          LD   (HL) A
OD06 18CC        0830          JR   CONT
OD08 EE80        0840          XOR  80         ; RETURN TO CURSOR
OD0A 2834        0850          JR   Z START    ; INVERT BIT 7
OD0C FE10        0860          CP   10         ; IF ZERO, START
OD0E 28CF        0870          JR   Z CURO     ; DEFAULT?
OD10 4F          0880          LD   C A        ; PUT ENTRY INTO EC
OD11 13          0890          INC  DE
OD12 1A          0900          LD   A (DE)
OD13 47          0910          LD   B A
OD14 2B          0920          ADD  HL BC      ; NEW CURSOR ADDRESS
OD15 7D          0930          LD   A L        ; THE NEXT SECTION
OD16 E63F        0940          AND  3F         ; IMPLEMENTS SCREEN
OD18 FE3A        0950          CP   3A         ; WRAPAROUND
OD1A 2005        0960          JR   NZ EDGE1
OD1C 7D          0970 EDGE0    LD   A L
OD1D EE30        0980          XOR  30
OD1F 6F          0990          LD   L A
OD20 188C        1000 STEP     JR   CURO
OD22 FE09        1010 EDGE1    CP   9
OD24 28F6        1020          JR   Z EDGE0
OD26 7C          1030          LD   A H
OD27 FE07        1040          CP   7
OD29 2006        1050          JR   NZ EDGE2
OD2B 2608        1060          LD   H 8
OD2D CBB5        1070 EDGRES   RES  6 L
OD2F 18EF        1080          JR   STEP
OD31 E5          1090 EDGE2    PUSH HL
OD32 01BA0B      1100          LD   BC 0BBA    ; CLEAR THE
OD35 ED42        1110          SBC  HL BC      ; WORKSPACE
OD37 E1          1120          POP  HL
OD38 38F5        1130          JR   C EDGRES+2
OD3A 2608        1140          LD   H 8
OD3C CBBD        1150          RES  7 L
OD3E 18ED        1160          JR   EDGRES
OD40 217B10      1170 START    LD   HL STORE
OD43 117C10      1180          LD   DE STORE+1
```

```
                0010 : A LIFE PROGRAM FOR NASCOM 1 AND 2
0C80            0020         ORG  £0C80
0C80 210A08     0030 CLRSCR  LD   HL £80A      : START OF SCREEN RAM
0C83 0630       0040         LD   B £30        : LINE LENGTH
0C85 36C0       0050 CLL     LD   (HL) £C0     : 'GRAPHICS' SPACE
0C87 2C         0060         INC  L            : CLEAR FIRST LINE
0C88 10FB       0070         DJNZ CLL
0C8A 2E0A       0080         LD   L £A         : COPY FIRST LINE
0C8C 114A08     0090         LD   DE £84A      : TO REST OF SCREEN
0C8F 017003     0100         LD   BC £370
0C92 EDB0       0110         LDIR
0C94 3E01       0120         LD   A 1          : 1 MARKS END OF SCREEN
0C96 12         0130         LD   (DE) A
0C97 215810     0140         LD   HL TITLE     : PUT TITLE ON TOP LINE
0C9A 0E23       0150         LD   C 35
0C9C 1ECF       0160         LD   E £CF
0C9E EDB0       0170         LDIR
0CA0 21991A     0180         LD   HL STORE+2590: N.(N-L+1)+(N-M)/2
0CA3 222810     0190         LD   (DSPADD) HL  : INITIALISE DISPLAY
0CA6 21280E     0200         LD   HL £E28      : INTIALISE Y AND X
0CA9 222A10     0210         LD   (POSY) HL    : COORDINATES
0CAC 21E209     0220 CURSOR  LD   HL £9E2      : SCREEN CENTRE
0CAF E5         0230 CURO    PUSH HL           : CONVERT CURSOR
0CB0 7D         0240         LD   A L          : ADDRESS TO XY
0CB1 E6C0       0250         AND  £C0          : COORDINATES
0CB3 C606       0260         ADD  A,£06
0CB5 94         0270         SUB  H
0CB6 07         0280         RLCA
0CB7 07         0290         RLCA
0CB8 EE03       0300         XOR  3
0CBA 3D         0310         DEC  A
0CBB 47         0320         LD   B A
0CBC 87         0330         ADD  A,A
0CBD 80         0340         ADD  A,B
0CBE 4F         0350         LD   C A
0CBF 7D         0360         LD   A L
0CC0 E63F       0370         AND  £3F
0CC2 D60A       0380         SUB  £0A
0CC4 87         0390         ADD  A,A
0CC5 47         0400         LD   B A
0CC6 CD0010     0410         CALL COORD        : DISPLAY COORDINATES
0CC9 E1         0420         POP  HL
0CCA 4E         0430 CUR1    LD   C (HL)       : SAVE CHARACTER IN C
0CCB 165F       0440         LD   D £5F        : CURSOR CHARACTER
0CCD 7E         0450 CUR2    LD   A (HL)       : FLASH CURSOR
0CCE 72         0460         LD   (HL) D
0CCF 57         0470         LD   D A
0CD0 061E       0480         LD   B 30         : DELAY LOOP
0CD2 DF         0490 CUR3    DEFB £DF 62 0     : SCAN KEYBOARD
0CD5 3805       0500         JR   C WRITE      : IF KEY PRESSED,JUMP
0CD7 FF         0510         RST  56
0CD8 10F8       0520         DJNZ CUR3         : CONTINUE DELAY LOOP
0CDA 18F1       0530         JR   CUR2         : REPEAT CURSOR FLASH
0CDC 71         0540 WRITE   LD   (HL) C       : REPLACE CHARACTER
0CDD EB         0550         EX   DE HL        : CURSOR ADRESS IN DE
0CDE 213210     0560         LD   HL TSTART-3  : TABLE SEARCH ROUTINE
0CE1 325610     0570         LD   (TEND) A     : SEARCH DEFAULT
0CE4 23         0580 WRT1    INC  HL           : INCREMENT TO
```

```
0D46 01401F     1190         LD   BC 8000
0D49 77         1200         LD   (HL) A
0D4A EDB0       1210         LDIR
0D4C 3C         1220         INC  A            : 1 MARKS END OF
0D4D 12         1230         LD   (DE) A       : WORKSPACE
0D4E 210A08     1240         LD   HL £80A      : START OF SCREEN
0D51 118D00     1250         LD   DE 141       : 2.N+(N-M)/2
0D54 DD2A2810   1260         LD   IX (DSPADD)  : POSITION OF SCREEN
0D58 7E         1270 COPY    LD   A (HL)       : COPY TO WORKSPACE
0D59 3D         1280         DEC  A            : 1 MARKS END OF SCREEN
0D5A 2843       1290         JR   Z COUNT
0D5C 3C         1300         INC  A            : 0 MARKS END OF LINE
0D5D 2837       1310         JR   Z NXTLIN
0D5F CB5F       1320         BIT  3 A          : MORE UNRAVELLING
0D61 2804       1330         JR   Z COPY0
0D63 EE04       1340         XOR  4
0D65 1806       1350         JR   COPY1
0D67 CB57       1360 COPY0   BIT  2,A
0D69 2802       1370         JR   Z COPY1
0D6B EE20       1380         XOR  £20
0D6D 1F         1390 COPY1   RRA                : COPY EACH PIXEL TO
0D6E DDCBC11E   1400         RR   (IX-63)       : BIT 7 OR BIT 3
0D72 1F         1410         RRA                : OF THE APROPRIATE BYTE
0D73 DDCB001E   1420         RR   (IX)          : IN THE WORKSPACE
0D77 1F         1430         RRA
0D78 3004       1440         JR   NC COPY1A
0D7A DDCB00DE   1450         SET  3 (IX)
0D7E 1F         1460 COPY1A  RRA
0D7F 3004       1470         JR   NC COPY1B
0D81 DDCB3FE6   1480         SET  4 (IX+63)    : NEXT RR CHANGES
0D85 1F         1490 COPY1B  RRA                : BIT 4 TO BIT 3
0D86 DDCB3F1E   1500         RR   (IX+63)
0D8A 1F         1510         RRA
0D8B 3004       1520         JR   NC COPY1C
0D8D DDCBC1DE   1530         SET  3 (IX-63)
0D91 DD23       1540 COPY1C  INC  IX            : GET NEXT SCREEN BYTE
0D93 23         1550         INC  HL
0D94 18C2       1560         JR   COPY
0D96 23         1570 NXTLIN  INC  HL            : AT END OF LINE
0D97 7E         1580         LD   A (HL)       : SKIP OVER NULLS
0D98 A7         1590         AND  A
0D99 28FB       1600         JR   Z NXTLIN
0D9B DD19       1610         ADD  IX DE
0D9D 18B9       1620         JR   COPY
0D9F 21BA10     1630 COUNT   LD   HL STORE+63  : COUNT NEIGHBOURS
0DA2 AF         1640 CT0     XOR  A
0DA3 23         1650 CT1     INC  HL           : IF THE CELL IS EMPTY
0DA4 BE         1660         CP   (HL)         : IGNORE IT
0DA5 D2A30D     1670         JP   NC CT1
0DA8 B7         1680         OR   A
0DA9 11BB2F     1690         LD   DE STREND    : ARE YOU AT THE END
0DAC ED52       1700         SBC  HL DE        : OF THE WORKSPACE?
0DAE 19         1710         ADD  HL DE
0DAF 3047       1720         JR   NC OFTEST    : IF SO, TEST OVERFLOW
0DB1 113F00     1730         LD   DE 63
0DB4 CB7E       1740         BIT  7 (HL)       : IS BIT 7 SET?
0DB6 2814       1750         JR   Z THREE      : IF NOT, SKIP TO BIT 3
0DB8 34         1760         INC  (HL)         : ADD 1 TO ALL THE
0DB9 19         1770         ADD  HL DE        : ADJACENT NIBBLES
0DBA 7E         1780         LD   A (HL)
```

```
0DBB C611      1790          ADD   A, £11
0DBD 77        1800          LD    (HL) A
0DBE 2B        1810          DEC   HL
0DBF 34        1820          INC   (HL)
0DC0 ED52      1830          SBC   HL DE
0DC2 34        1840          INC   (HL)
0DC3 ED52      1850          SBC   HL DE
0DC5 34        1860          INC   (HL)
0DC6 23        1870          INC   HL
0DC7 7E        1880          LD    A (HL)
0DC8 C611      1890          ADD   A,£11
0DCA 77        1900          LD    (HL) A
0DCB 19        1910          ADD   HL DE
0DCC CB5E      1920 THREE    BIT   3 (HL)     : NOW TEST BIT 3
0DCE 28D2      1930          JR    Z CTO      : IF ZERO, CONTNUE SCAN
0DD0 CB56      1940          BIT   2 (HL)     : IF BIT 2 IS SET
0DD2 2802      1950          JR    Z TH1      : RESET BIT 3 TO PREVENT
0DD4 CB9E      1960          RES   3 (HL)     : O/FLOW TO HIGH NIBBLE
0DD6 7E        1970 TH1      LD    A (HL)     : ADD 1 TO ALL
0DD7 C610      1980          ADD   A,£10      : ADJACENT NIBBLES
0DD9 77        1990          LD    (HL) A
0DDA 19        2000          ADD   HL DE
0DDB 7E        2010          LD    A (HL)
0DDC C611      2020          ADD   A,£11
0DDE 77        2030          LD    (HL) A
0DDF 23        2040          INC   HL
0DE0 7E        2050          LD    A (HL)
0DE1 C610      2060          ADD   A,£10
0DE3 77        2070          LD    (HL) A
0DE4 ED52      2080          SBC   HL DE
0DE6 7E        2090          LD    A (HL)
0DE7 C610      2100          ADD   A,£10
0DE9 77        2110          LD    (HL) A
0DEA ED52      2120          SBC   HL DE
0DEC 7E        2130          LD    A (HL)
0DED C610      2140          ADD   A,£10
0DEF 77        2150          LD    (HL) A
0DF0 2B        2160          DEC   HL
0DF1 7E        2170          LD    A (HL)
0DF2 C611      2180          ADD   A,£11
0DF4 77        2190          LD    (HL) A
0DF5 19        2200          ADD   HL DE
0DF6 18AA      2210          JR    CTO        : CONTINUE SCAN
0DF8 3601      2220 OFTEST   LD    (HL) 1     : TEST FOR OVERFLOW
0DFA 217B10    2230          LD    HL STORE   : AT EDGE OF WORKSPACE
0DFD 113F00    2240          LD    DE 63
0E00 AF        2250          XOR   A
0E01 CDC40F    2260          CALL  CHECK      : TEST TOP EDGE
0E04 067D      2270          LD    B 125
0E06 19        2280 OFT1     ADD   HL DE      : TEST VERTICAL EDGE
0E07 BE        2290          CP    (HL)       : ARE ALL CELLS EMPTY?
0E08 2820      2300          JR    Z OFT4     : IF SO, CONTINUE
0E0A D5        2310          PUSH  DE         : IF NOT, THEN
0E0B 1E01      2320          LD    E 1        : TEST FOR A GLIDER
0E0D DD213210  2330          LD    IX GTAB3
0E11 7E        2340          LD    A (HL)
0E12 FE03      2350          CP    3
0E14 2809      2360          JR    Z OFT2
0E16 11FFFF    2370          LD    DE -1
0E19 DD212F10  2380          LD    IX GTAB2
0E1D FE30      2390          CP    £30
0E1F 3E00      2400 OFT2     LD    A 0        : CLEAR OVERFLOW
0E21 77        2410          LD    (HL) A
0E22 2005      2420          JR    NZ OFT3
0E24 0EFF      2430          LD    C £FF
0E26 CD910F    2440          CALL  GLT1       : TEST FOR GLIDER
0E29 D1        2450 OFT3     POP   DE
0E2A 10DA      2460 OFT4     DJNZ  OFT1
0E2C 11C1FF    2470          LD    DE -63     : CHECK BOTTOM EDGE
0E2F CDC40F    2480          CALL  CHECK
0E32 21BA10    2490          LD    HL STORE+63 : WORKSPACE SCAN
0E35 48                      LD    C B        : SET BC TO ZERO
0E36 C5        2510          PUSH  BC
0E37 FDE1      2520          POP   IY         : SET IY TO ZERO
0E39 23        2530 NXTGEN   INC   HL
0E3A BE        2540 NG1      CP    (HL)
0E3B CA390E    2550          JP    Z NXTGEN   : IGNORE EMPTY CELLS
0E3E 11B62F    2560          LD    DE STREND  : CHECK FOR END OF
0E41 B7        2570          OR    A          : WORKSPACE
0E42 ED52      2580          SBC   HL DE
0E44 19        2590          ADD   HL DE
0E45 300A      2600          JR    NC NGEND,  : JUMP IF END
0E47 CD000F    2610          CALL  EVOLVE     : USE EVOLVE ROUTINE
0E4A CD000F    2620          CALL  EVOLVE     : FOR EACH NIBBLE
0E4D ED6F      2630          RLD              : RESTORE ORIGINAL BYTE
0E4F 18E8      2640          JR    NXTGEN
0E51 CB18      2650 NGEND    RR    B          : ANY CHANGE IN PATTERN?
0E53 380F      2660          JR    C NG0      : IF SO, JUMP
0E55 211C10    2670          LD    HL NCLAB   : ADDRESS OF 'STOP'
0E58 CB18      2680          RR    B          : HAS THE PATTERN DIED?
0E5A 3303      2690          JR    C PROMPT   : IF NOT, JUMP
0E5C 212010    2700          LD    HL DOLAB   : ADDRESS OF 'DIES'
0E5F CDB70F    2710 PROMPT   CALL  PRINT      : PRINT COMMENT
0E62 182C      2720          JR    KBD        : WAIT FOR INSTRUCTION
0E64 CDCE0F    2730 NG0      CALL  GENNUM     : GENERATION COUNT
0E67 CD1F0F    2740          CALL  DISPLA     : DISPLAY PATTERN
0E6A 21E70B    2750          LD    HL £BE7    : CLEAR POPULATION COUNT
0E6D 0603      2760          LD    B 3
0E6F 3620      2770 NGLD     LD    (HL) £20
0E71 2C        2780          INC   L
0E72 10FB      2790          DJNZ  NGLD
0E74 3630      2800          LD    (HL) '0
0E76 FDE5      2810          PUSH  IY         : PUT POPULATION COUNT
0E78 C1        2820          POP   BC         : INTO BC
0E79 78        2830 POPNUM   LD    A B        : COUNT UNTIL BC=0
0E7A B1        2840          OR    C
0E7B 2808      2850          JR    Z BREAK    : JUMP
0E7D 2EEA      2860          LD    L £EA
0E7F CDD1 F    2870          CALL  ASCNUM
0E82 0B        2880          DEC   BC
0E83 18F4      2890          JR    POPNUM
0E85 21080C    2900 BREAK    LD    HL £C08    : IS THE SPACE BAR
0E88 DF        2910          DEFB  £DF 62 00  : DEPRESED?
0E8B CB66      2920          BIT   4 (HL)
0E8D CA9F0D    2930          JP    Z COUNT    : CONTINUE IF NOT
0E90 CF        2940 KBD      DEFB  £CF 0 0    : NEXT INSTRUCTION?
0E93 00        2950          NOP
0E94 FE52      2960          CP    'R         : RESTART, CLEAR SCREEN
0E96 CA800C    2970          JP    Z CLRSCR
0E99 FE4D      2980          CP    'M         : RESTART, BUT LEAVE
```

```
0E9B CAAC0C  2990          JP   Z CURSOR     : PATTERN ON SCREEN
0E9E FE20    3000          CP   £20
0EA0 CA9F0D  3010          JP   Z COUNT
0EA3 FE14    3020          CP   £14         : PAN DOWN (PATTERN
0EA5 200E    3030          JR   NZ DOWN     : MOVES UP)
0EA7 3A2A10  3040          LD   A (POSY)    : STOP WHEN YOU
0EAA FE50    3050          CP   80          : GET TO THE EDGE
0EAC 28E2    3060          JR   Z KBD
0EAE 11C1FF  3070          LD   DE -63
0EB1 C601    3080          ADD  A,1
0EB3 180F    3090          JR   D1
0EB5 FE13    3100 DOWN     CP   £13         : PAN UP
0EB7 2010    3110          JR   NZ LEFT
0EB9 3A2A10  3120          LD   A (POSY)
0EBC A7      3130          AND  A           : STOP AT THE EDGE
0EBD 28D1    3140          JR   Z KBD
0EBF 113F00  3150          LD   DE 63
0EC2 D601    3160          SUB  1
0EC4 322A10  3170 D1       LD   (POSY) A
0EC7 1824    3180          JR   SHIFT
0EC9 FE12    3190 LEFT     CP   £12         : PAN RIGHT (PATTERN
0ECB 200D    3200          JR   NZ RIGHT    : MOVES LEFT)
0ECD 3A2B10  3210          LD   A (POSX)
0ED0 A7      3220          AND  A           : STOP AT THE EDGE
0ED1 28BD    3230          JR   Z KBD
0ED3 11FFFF  3240          LD   DE -1
0ED6 D602    3250          SUB  2
0ED8 1810    3260          JR   R1
0EDA FE11    3270 RIGHT    CP   £11         : PAN LEFT
0EDC 20B2    3280          JR   NZ KBD
0EDE 3A2B10  3290          LD   A (POSX)
0EE1 FE1E    3300          CP   30          : STOP AT EDGE
0EE3 28AB    3310          JR   Z KBD
0EE5 110100  3320          LD   DE 1
0EE8 C602    3330          ADD  A,2
0EEA 322B10  3340 R1       LD   (POSX) A
0EED 2A2810  3350 SHIFT    LD   HL (DSPADD) : GET NEW ADDRESS
0EF0 19      3360          ADD  HL DE       : OF DISPLAY
0EF1 222810  3370          LD   (DSPADD) HL
0EF4 CD1F0F  3380          CALL DISPLA      : PUT NEW POSITION
0EF7 ED4B2A10 3390         LD   BC (POSY)   : ON THE SCREEN
0EFB CD0010  3400          CALL COORD       : DISPLAY COORDINATES
0EFE 1890    3410          JR   KBD         : RETURN FOR NEXT INSTR.
0F00 ED6F    3420 EVOLVE   RLD              : GET TOP NIBBLE
0F02 FE03    3430          CP   3           : IS IT 3?
0F04 2810    3440          JR   Z EV2       : IF SO, CELL BORN
0F06 FE08    3450          CP   8           : LESS THAN 8?
0F08 380A    3460          JR   C EV1       : EMPTY CELL,NO CHANGE
0F0A FE0A    3470          CP   £A          : LESS THAN 10
0F0C 3804    3480          JR   C EV0       : IF SO, CELL DIES
0F0E FE0C    3490          CP   £C          : BETWEEN 10 AND 12
0F10 3806    3500          JR   C EV3       : IF SO, REMAINS ALIVE
0F12 CBC0    3510 EV0      SET  0 B         : BIT 0,B FLAGS CHANGE
0F14 AF      3520 EV1      XOR  A           : CLEAR CELL
0F15 C9      3530          RET
0F16 CBC0    3540 EV2      SET  0 B         : FLAG CHANGE
0F18 3E08    3550 EV3      LD   A 8         : LIVE CELL
0F1A CBC8    3560          SET  1 B         : FLAG CELL ALIVE
0F1C FD23    3570          INC  IY          : POPULATION COUNT IN IY
0F1E C9      3580          RET
```

```
0F1F DD2A2810 3590 DISPLA  LD   IX (DSPADD) : GET ADDRESS
0F23 210A08   3600         LD   HL £80A     : START OF SCREEN
0F26 0E0F     3610         LD   C 15        : NUMBER OF LINES
0F28 0630     3620 DIS1    LD   B 48        : NUMBER OF CHARACTERS
0F2A 3EC0     3630 DIS2    LD   A £C0       : GRAPHICS SPACE
0F2C 1E00     3640         LD   E 0
0F2E DDCBC15E 3650         BIT  3 (IX-63)   : MAP THE WORKSPACE
0F32 2802     3660         JR   Z DIS2A     : NIBBLES TO THE
0F34 CBEF     3670         SET  5 A         : SCREEN PIXELS
0F36 DDCB3F7E 3680 DIS2A   BIT  7 (IX+63)   : THE TWIDDLY BITS
0F3A 2802     3690         JR   Z DIS2B     : ARE  JUST MORE
0F3C CBE7     3700         SET  4 A         : UNRAVELLING
0F3E DDCB3F5E 3710 DIS2B   BIT  3 (IX+63)
0F42 2804     3720         JR   Z DIS2C
0F44 CBDF     3730         SET  3 A
0F46 CBD3     3740         SET  2 E
0F48 DDCB005E 3750 DIS2C   BIT  3 (IX)
0F4C 2802     3760         JR   Z DIS2D
0F4E CBD7     3770         SET  2 A
0F50 AB       3780 DIS2D   XOR  E
0F51 DDCB007E 3790         BIT  7 (IX)
0F55 2802     3800         JR   Z DIS2E
0F57 CBCF     3810         SET  1 A
0F59 DDCBC17E 3820 DIS2E   BIT  7 (IX-63)
0F5D 2802     3830         JR   Z DIS2F
0F5F CBC7     3840         SET  0 A
0F61 57       3850 DIS2F   LD   D A
0F62 E60C     3860         AND  £C
0F64 FE04     3870         CP   4
0F66 7A       3880         LD   A D
0F67 2002     3890         JR   NZ NEXT
0F69 EE20     3900         XOR  £20
0F6B DD23     3910 NEXT    INC  IX
0F6D 77       3920         LD   (HL) A
0F6E 2C       3930         INC  L
0F6F 10B9     3940         DJNZ DIS2
0F71 118D00   3950         LD   DE 141
0F74 DD19     3960         ADD  IX DE
0F76 111000   3970         LD   DE 16
0F79 19       3980         ADD  HL DE
0F7A 0D       3990         DEC  C
0F7B 20AB     4000         JR   NZ DIS1
0F7D C9       4010         RET
0F7E 7E       4020 GLTEST  LD   A (HL)      : TEST FOR GLIDER
0F7F 0E0F     4030         LD   C £F
0F81 FE23     4040         CP   £23
0F83 2804     4050         JR   Z GLT0
0F85 0EF0     4060         LD   C £F0
0F87 FE32     4070         CP   £32
0F89 3E00     4080 GLT0    LD   A 0
0F8B 77       4090         LD   (HL) A
0F8C C0       4100         RET  NZ
0F8D DD212C10 4110         LD   IX GTAB1
0F91 E5       4120 GLT1    PUSH HL
0F92 C5       4130         PUSH BC
0F93 0603     4140         LD   B 3
0F95 19       4150 GLT2    ADD  HL DE
0F96 7E       4160         LD   A (HL)
0F97 3600     4170         LD   (HL) 0
0F99 DDAE00   4180         XOR  (IX)        : IF IT'S NOT A GLIDER
```

```
0F9C A1       4190          AND  C            : IT MUST BE
0F9D 200C     4200          JR   NZ OVRFLO    : AN OVERFLOW
0F9F DD23     4210          INC  IX
0FA1 10F2     4220          DJNZ GLT2
0FA3 21D90B   4230          LD   HL £BD9      : COUNT ONE GLIDER
0FA6 CDD10F   4240          CALL ASCNUM
0FA9 2008     4250          JR   NZ END       : RETURN VIA END
0FAB 212410   4260 OVRFLO   LD   HL OFLAB     : PRINT 'O/FL'
0FAE D5       4270          PUSH DE
0FAF CDB70F   4280          CALL PRINT
0FB2 D1       4290          POP  DE
0FB3 C1       4300 END      POP  BC
0FB4 E1       4310          POP  HL
0FB5 AF       4320          XOR  A
0FB6 C9       4330          RET
0FB7 010400   4340 PRINT    LD   BC 4         : PRINT 4 LETTER REM
0FBA 11CF0B   4350 PR1      LD   DE £BCF
0FBD 1A       4360          LD   A (DE)
0FBE FE4F     4370          CP   'O
0FC0 C8       4380          RET  Z
0FC1 EDB0     4390          LDIR
0FC3 C9       4400          RET
0FC4 063F     4410 CHECK    LD   B 63         : SCANS 63 EDGE CELLS
0FC6 23       4420 CH1      INC  HL
0FC7 BE       4430          CP   (HL)
0FC8 C47E0F   4440          CALL NZ GLTEST
0FCB 10F9     4450          DJNZ CH1
0FCD C9       4460          RET
0FCE 21E20B   4470 GENNUM   LD   HL £BE2      : COUNT ONE GEN.
0FD1 7E       4480 ASCNUM   LD   A (HL)       : ASCII COUNT ROUTINE
0FD2 F630     4490          OR   'O           : CONVERT SPACE TO ZERO
0FD4 3C       4500          INC  A            : INCREMENT
0FD5 77       4510          LD   (HL) A
0FD6 FE3A     4520          CP   '9+1         : GREATER THAN 9?
0FD8 C0       4530          RET  NZ           : IF NOT, RETURN
0FD9 3630     4540          LD   (HL) 'O      : PRINT A ZERO
0FDB 2D       4550          DEC  L            : CARRY ONE
0FDC 18F3     4560          JR   ASCNUM
1000          4570          ORG  £1000
1000 21EE0B   4580 COORD    LD   HL £BEE      : £1000 TO LEAVE
1003 CD0910   4590          CALL CRD1         : SPACE FOR STACK
1006 41       4600          LD   B C
1007 2EF3     4610          LD   L £F3
1009 3620     4620 CRD1     LD   (HL) £20     : CLEAR COORDINATES
100B 2C       4630          INC  L
100C 3630     4640          LD   (HL) 'O
100E 04       4650          INC  B
100F 05       4660 CRD2     DEC  B            : PRINT COORDINATES
1010 C8       4670          RET  Z
1011 E5       4680          PUSH HL
1012 CDD10F   4690          CALL ASCNUM
1015 E1       4700          POP  HL
1016 18F7     4710          JR   CRD2
1018 20       4720 OSCLAB   DEFM / OSC/       : TABLE OF REMARKS
101C 53       4730 NOLAB    DEFM /STOP/
1020 44       4740 DOLAB    DEFM /DIES/
1024 4F       4750 OFLAB    DEFM -O/FL-
0002          4760 DSPADD   DEFS 2            : CURRENT DISPLAY ADDRESS
0001          4770 POSY     DEFS 1            : Y COORDINATE
0001          4780 POSX     DEFS 1            : X COORDINATE
102C BB       4790 GTAB1    DEFB £BB £55 £99  : TABLE FOR
102F 5B       4800 GTAB2    DEFB £5B £19 00   : GLIDER TEST
1032 B5       4810 GTAB3    DEFB £B5 £91 00
1035 5A       4820 TSTART   DEFB 'Z 0 £10     : PIXEL CONTROL
1038 58       4830          DEFB 'X 0 £C
103B 41       4840          DEFB 'A 0 2
103E 53       4850          DEFB 'S 0 4
1041 51       4860          DEFB 'Q 0 1
1044 57       4870          DEFB 'W 0 £20
1047 11       4880          DEFB £11 £7F £FF  : CURSOR CONTROL
104A 12       4890          DEFB £12 £81 0
104D 14       4900          DEFB £14 £C0 0
1050 13       4910          DEFB £13 £40 £FF
1053 20       4920          DEFB £20 £80 0    : SPACE BAR
1056 00       4930 TEND     DEFB 0 £90        : DEFAULT POSITION
1058 4C       4940 TITLE    DEFM /LIFE  GL /  : GLIDER COUNT
1061 20       4950          DEFM /  :GEN   /  : GENERATION NUMBER
106A 20       4960          DEFM /  :POP   /  : TOTAL POPULATION
1073 20       4970          DEFM / :X    Y/   : XY COORDINATES
1F41          4980 STORE    DEFS 8001         : WORKSPACE
2FBB          4990 STREND   EQU  STORE+8000
```

MODS. CHANGE FOR T4, BBUG

1A	OCD2	CD				
1A	OCD2	CD	69	00	LINE490	CALL £69
	OE88	CD	69	00	LINE2910	CALL £69
	OE90	CD	3E	00	LINE2940	CALL £3E
1B	304D	CD	3E	00	LINE560	CALL £3E

CHANGES FOR DIFFERENT CURSOR

2A (a) PUT NEW KEYS IN TABLE, LINES
4880 - 4910

DEFB "L",";","."P @ £1047,£104A,£104D,£1050

(b) ALSO @	OEA4".	Line	3020
	OEB6"P		3100
	OECA";		3190
	OEDA "L		3270

2B	30B5 "L	1030	
	30BA ";"	1060	
	30BF "P	1090	
	30C5 ".	1130	

CHANGES FOR BITS AND P.C.'S GRAPHICS.

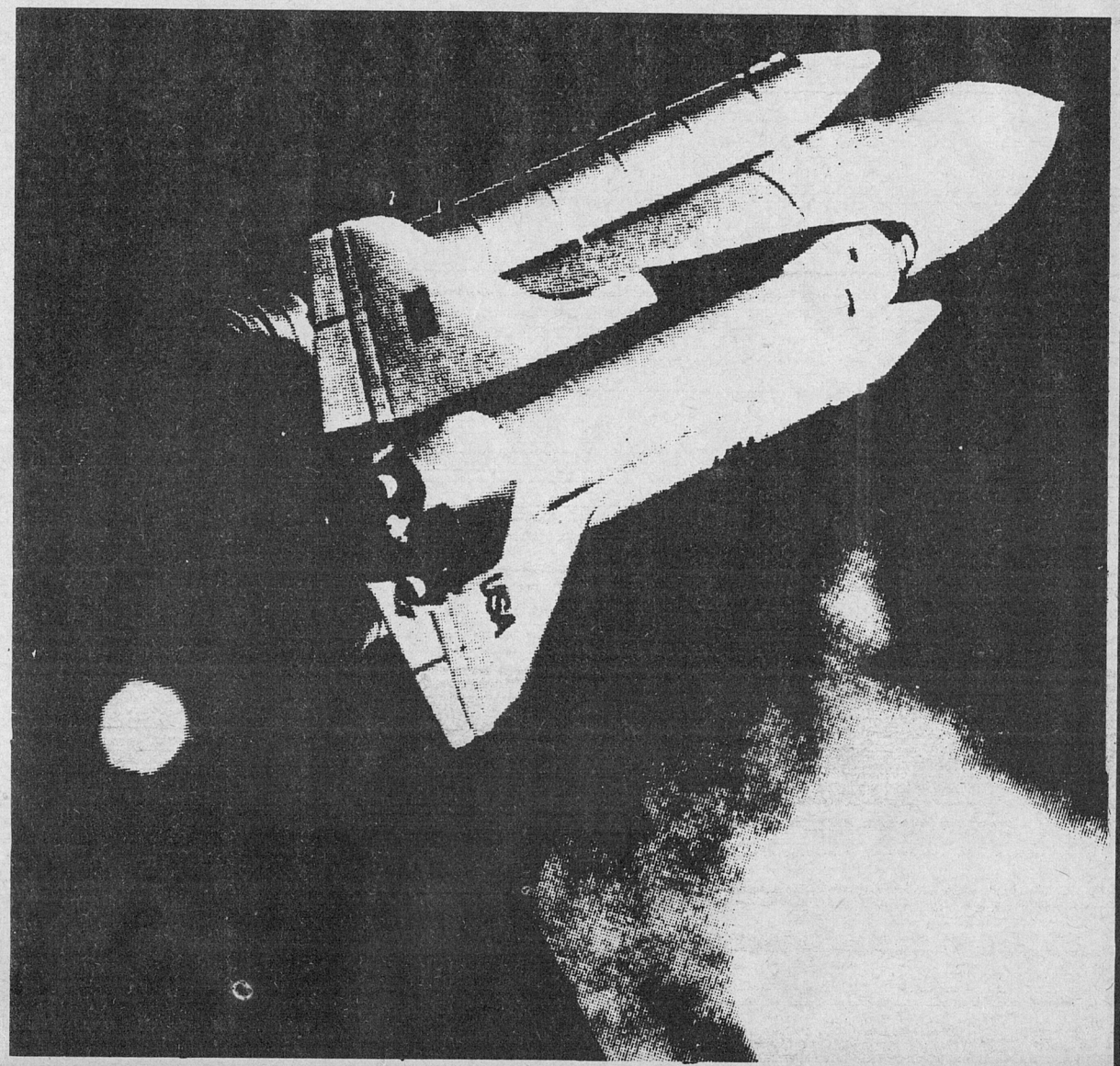
3A	0C85	36A0	LINE 50	LD HL,£A0
	OCF2	1810	LINE 700	JR XORA

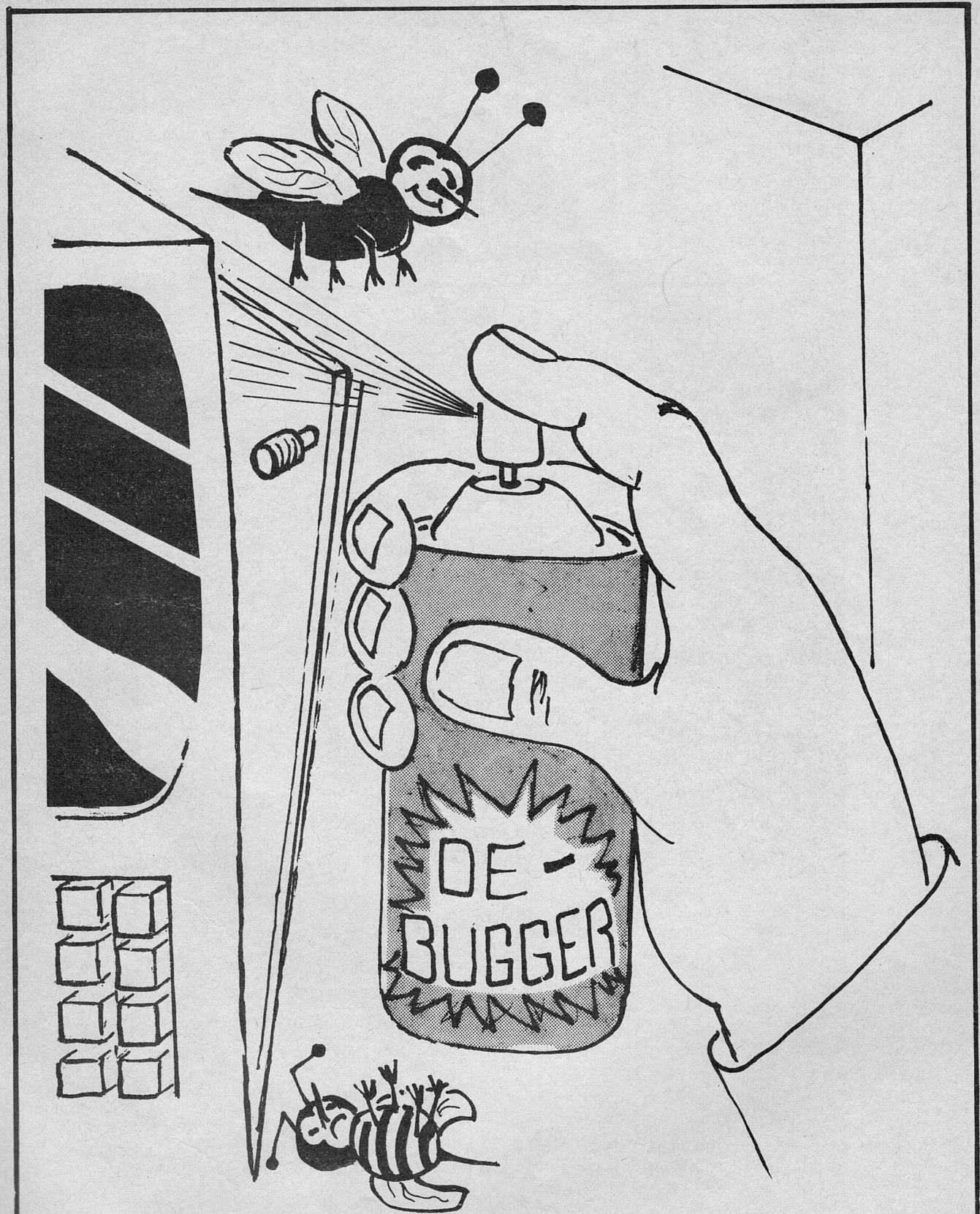
CHANGE TABLE TO 1, 2, 4, 8, £10, £5F
4820 - 4870

3B	3119	1810	LINE 700	JR PXORA

```
                0010 : PATCH 1A
                0020 : CHANGES TO LIFE PROGRAM FOR
                0030 : BITS AND P.C.S GRAPHICS BOARD
                0040          ORG  £D5F          : START OF PATCH
OD5F CB77       0050          BIT  6 A           : TEST VICEO INVERT BIT
OD61 2806       0060          JR   Z COPY1       : IF NOT SET, SKIP
OD63 EE1F       0070          XOR  £1F           : INVERT OTHER BITS
OD65 DDCBC1DE   0080          SET  3 (IX-63)     : NOW MAP SCREEN PIXELS
OD69 1F         0090 COPY1    RRA                : TO APPROPRIATE BITS
OD6A DDCB3F1E   0100          RR   (IX+63)       : IN THE WORKSPACE
OD6E 1F         0110          RRA
OD6F 3004       0120          JR   NC COPY1A
OD71 DDCB3FDE   0130          SET  3 (IX+63)
OD75 1F         0140 COPY1A   RRA
OD76 DDCB001E   0150          RR   (IX)
OD7A 1F         0160          RRA
OD7B 3004       0170          JR   NC COPY1B
OD7D DDCB00DE   0180          SET  3 (IX)
OD81 1F         0190 COPY1B   RRA
OD82 300D       0200          JR   NC COPY1C
OD84 DDCBC1FE   0210          SET  7 (IX-63)
OD88 1807       0220          JR   COPY1C        : BACK TO MAIN PROGRAM
                0230 : END OF PATCH 1A
                0240 :
                0250 : PATCH 1B
                0260 : MODIFICATIONS TO DISPLAY ROUTINE
                0270 : FOR BITS AND P.C.S BOARD
                0280          ORG  £F2A
OF2A 3EA0       0290 DIS2     LD   A £A0         : GRAPHICS SPACE
OF2C DDCBC17E   0300          BIT  7 (IX-63)     : COPY WORKSPACE TO
OF30 2802       0310          JR   Z DIS2A       : SCREEN PIXELS
OF32 CBE7       0320          SET  4 A
OF34 DDCB005E   0330 DIS2A    BIT  3 (IX)
OF38 2802       0340          JR   Z DIS2B
OF3A CBDF       0350          SET  3 A
OF3C DDCB007E   0360 DIS2B    BIT  7 (IX)
OF40 2802       0370          JR   Z DIS2C
OF42 CBD7       0380          SET  2 A
OF44 DDCB3F5E   0390 DIS2C    BIT  3 (IX+63)
OF48 2802       0400          JR   Z DIS2D
OF4A CBCF       0410          SET  1 A
OF4C DDCB3F7E   0420 DIS2C    BIT  7 (IX+63)
OF50 2802       0430          JR   Z DIS2E
OF52 CBC7       0440          SET  0 A
OF54 DDCBC15E   0450 DIS2E    BIT  3 (IX-63)     : VICEO INVERT BIT
OF58 2811       0460          JR   Z NEXT        : IF RESET, SKIP
OF5A EE5F       0470          XOR  £5F           : INVERT ALL BITS
OF5C 180D       0480          JR   NEXT          : BACK TO MAIN PROGRAM
                0490 : END OF PATCH 1B

                0010 : PATCH 2A
                0020 : CHANGES TO LIFE PROGRAM FOR
                0030 : USE ON A SYSTEM WITHOUT GRAPHICS
                0040          ORG  £D51
OD51 116600     0050          LD   DE 102
OD54 DD2A2810   0060          LD   IX (DSPADD)
OD58 7E         0070 COPY     LD   A (HL)
OD59 3D         0080          DEC  A
OD5A 2843       0090          JR   Z COUNT
OD5C 3C         0100          INC  A
```

```
0D5D 2837      0110          JR   Z NXTLIN
0D5F FE27      0120          CP   £27
0D61 2804      0130          JR   Z COPY1A
0D63 FE7C      0140          CP   £7C
0D65 2004      0150          JR   NZ COPY1B
0D67 DDCB00FE  0160 COPY1A   SET  7 (IX)
0D6B E608      0170 COPY1B   AND  8
0D6D 2804      0180          JR   Z COPY2
0D6F DDCB3FFE  0190          SET  7 (IX+63)
0D73 23        0200 COPY2    INC  HL
0D74 7E        0210          LD   A (HL)
0D75 FE27      0220          CP   £27
0D77 2804      0 30          JR   Z COPY2A
0D79 FE7C      0240          CP   £7C
0D7B 2004      0250          JR   NZ COPY2B
0D7D DDCB00DE  0260 COPY2A   SET  3 (IX)
0D81 E608      0270 COPY2B   AND  8
0D83 280C      0280          JR   Z COPY1C
0D85 DDCB3FDE  0290          SET  3 (IX+63)
0D89 1806      0300          JR   COPY1C
               0310 : END OF PATCH 2A
               0320 :
               0330 : PATCH 2B
OF1F           0340          ORG  £F1F
OF1F DD2A2B10  0350 DISPLA   LD   IX (DSPADD)
OF23 210A08    0360          LD   HL £80A
OF26 0E0F      0370          LD   C 15
OF28 0618      0380 DIS1     LD   B 24
OF2A 3E20      0390 DIS2     LD   A £20
OF2C DDCB007E  0400          BIT  7 (IX)
OF30 2802      0410          JR   Z DIS2A
OF32 3E27      0420          LD   A £27
OF34 DDCB3F7E  0430 DIS2A    BIT  7 (IX+63)
OF38 2806      0440          JR   Z DIS2B
OF3A 07        0450          RLCA
OF3B 07        0460          RLCA
OF3C 17        0470          RLA
OF3D 07        0480          RLCA
OF3E F62C      0490          OR   £2C
OF40 77        0500 DIS2B    LD   (HL) A
OF41 23        0510          INC  HL
OF42 3E20      0520          LD   A £20
OF44 DDCB005E  0530          BIT  3 (IX)
OF48 2802      0540          JR   Z DIS2C
OF4A 3E27      0550          LD   A £27
OF4C DDCB3F5E  0560 DIS2C    BIT  3 (IX+63)
OF50 2806      0570          JR   Z DIS2D
OF52 07        0580          RLCA
OF53 07        0590          RLCA
OF54 17        0600          RLA
OF 5 07        0610          RLCA
OF56 F62C      0620          OR   £2C
OF58 77        0630 DIS2D    LD   (HL) A
OF59 23        0640          INC  HL
OF5A DD23      0650          INC  IX
OF5C 10CC      0660          DJNZ DIS2
OF5E 111000    0670          LD   DE 16
OF61 19        0680          ADD  HL DE
OF62 115600    0690          LD   DE 102
OF65 DD19      0700          ADD  IX DE
OF67 0D        0710          DEC  C
OF68 20BE      0720          JR   NZ DIS1
OF6A C9        0730          RET
               0740 : END OF PATCH 2B
```





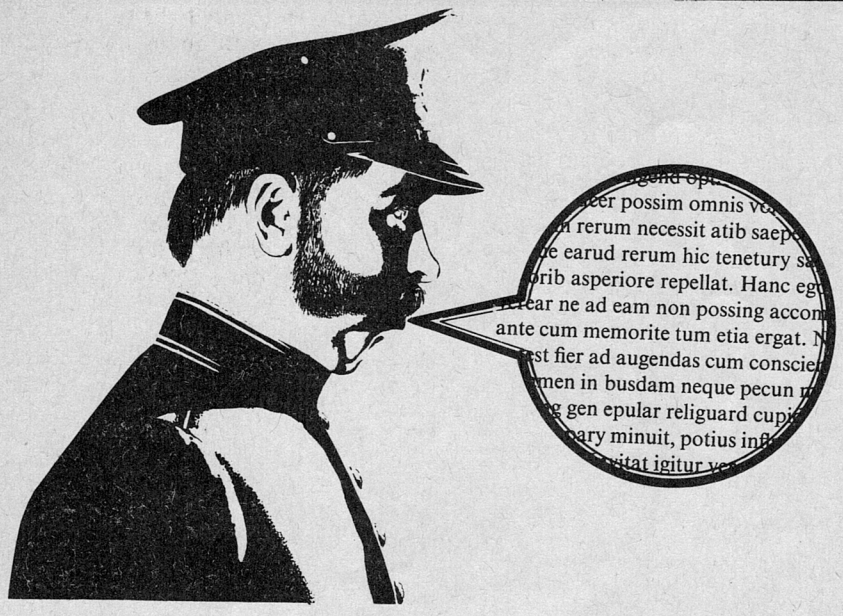
Dymanic De-Bug Tool!

6800 PROCESSOR PROFILE

BY DAVE STRAKER

Instr.	Oper'ds		
ABA		A-A+B	HNVZC
ADC	amdxe	a-a+M+C	HNVZC
ADD	amdxe	a-a+M	HNVZC
AND	amdxe	a-a M	NZ;V=0
ASL	axe	C-M(7);M(q)-M(q-1);M(0)-0	CNZ;V=CON
ASR	axe	C-M (0);M(p-1)-M(p)	CNZ;V=CON
BCC	r	if C=0 then P-EA	-
BCS	r	if C=1 then P-EA	-
BEQ	r	if Z=1 then P-EA	-
BGE	r	if N=V then P-EA	-
BGT	r	if Z=0 N=V then P-EA	-
BHI	r	if Z=0 C=0 then P-EA	-
BIT	a mdxe	if a=M then P-EA	HNVZC
BLE	r	If Z=1 N=V then P-EA	-
BLS	r	if C=1 Z=1 then P-EA	-
BLT	r	if N=V then P-EA	-
BMI	r	if N=1 then P-EA	-
BNE	r	if Z=0 then P-EA	-
BPL	r	if N=0 then P-EA	-
BRA	r	P-EA	-
BSR	r	P42;P-EA	-
BVC	r	if V=0 then P-EA	-
BVSr		if V=1 then P-EA	-
CBA		A-B	NVZC
CLC		C-0	C=0
CLI		I-0	I=0
CLR	axe	M-o	N=V=C=0Z=
CLV		V-0	V=0
CMP	a mdxe	a-M	NVZC
CDM	axe	M-M	Z NV NZ:V=
CPX	mdxe	X-M	NZC V ZNY
DAA		decimal adjust A	NZC V
DEC	axe	M -M -1	NVZ
DES		S - S - 1	-
DEX		X - X - 1	Z
EOR	a mdxe	a - a O M	NZ;V=0
INC	axe	M - M + 1	NVZ
INS		S - S + 1	-
INX		X - X + 1	Z
JMP	xe	P-EA	-
JSR	xe	P +(2 or 3)1; Ṗ-EA	-
LDA	a mdxe	a - M	NZ;V=0
LDS	mdxe	S-M	NZ;V=0
LDX	mdxe	X - M	NZ;V=C
LSR	axe	C-M(0);M(p-1)-M(p);M(7)-0	NZ;V=C
NEG	axe	M - 0-M	NVZ;C=L
NOP		no operation	-
ORA	a mdxe	a - a v M	NZ;V=0
PSH	a	(s) - a;s - s - 1 (=a)	-
PUL	a	S - S+1;a - (s) (=a)	-

```
0016:                           POP BUFFER      ;BUFFER ADDRESS
0016: 68                #       PLA
0017: 85 04             #       STA BUFFER
0019: 68                #       PLA
PAGE -   2  GETTEXT   FILE:AFOREIGN

001A: 85 05             #       STA BUFFER+1
001C: A9 52                     LDA #52         ;SET BAUD RATE AT 300 #11 FOR 1200
001E: 8D AEC0                   STA STATUS
0021:
0021: A9 00                     LDA #0
0023: 85 06                     STA NBYTES
0025: 85 07                     STA NBYTES+1    ;ZERO TEMPORARY ADDRESSES
0027: 85 08                     STA CHARS
0029: 85 0C                     STA SKCOUNT
002B:
002B: AD 00C0           WAIT    LDA KEYBOARD
002E: C9 8D                     CMP #8D         ;<CR> PRESSED ?
0030: F0**                      BEQ DONE        ;YES THEN QUIT
0032: AD AEC0                   LDA STATUS      ;TEST ACIA STATUS
0035: 6A                        ROR A
0036: 90F3                      BCC WAIT        ;UNTIL CHARACTER THERE
0038:
0038: AD AFC0                   LDA DATA        ;GET DATA
003B: 29 7F                     AND #7F         ;STRIP BIT 7
003D: C9 20                     CMP #20         ;IS IT A CONTROL CHARACTER ?
003F: B0**                      BCS STORIT      ;NO THEN STORE IT IN BUFFER
0041: C9 0D                     CMP #0D         ;IS IT A <CR> ?
0043: F0**                      BEQ CR          ;YES THEN ADD A LINE
0045: C9 09                     CMP #9          ;IS IT A TAB ?
0047: D0E2                      BNE WAIT        ;NO GET ANOTTHER CHARACTER
0049:
0049: A9 20             SPACES  LDA #20         ;IF YES THEN STORE SPACES
004B: 20 ****                   JSR STOREONE
004E: F0**                      BEQ DONE        ;BUT QUIT IF BUFFER FULL
0050: A5 08                     LDA CHARS
0052: 29 07                     AND #7
0054: D0F3                      BNE SPACES
0056: F0D3                      BEQ WAIT        ;GET ANOTHER CHARACTER IF DONE
0058:
0043* 00
0058: A0 00             CR      LDY #0
005A: 84 08                     STY CHARS       ;RESET # CHARS THIS LINE
005C: A4 0A                     LDY SKBYTES
005E: 84 0C                     STY SKCOUNT     ;RESET BYTE SKIP COUNTER
0060: 20 ****                   JSR DOIT        ;STORE <CR> IN BUFFER
0063: F0**                      BEQ DONE        ;QUIT IF BUFFER FULL
0065: D0C4                      BNE WAIT        ;NEXT CHAR IF NOT
0067:
003F* 00
0067: 20 ****           STORIT  JSR STOREONE    ;PUT CHAR IN BUFFER
006A: F0**                      BEQ DONE        ;QUIT IF BUFFER FULL
006C: D0BD                      BNE WAIT        ;ELSE NEXT CHAR
006E:
0068* 6E00
004C* 6E00
006E: E6 08             STOREONE INC CHARS      ;BUMP LOGICAL COUNT CHARS THIS LINE
0070: A4 0C                     LDY SKCOUNT     ;FINISHED YET ?
0072: F0**                      BEQ DOIT        ;YES,STORE IN BUFFER
0074: C6 0C                     DEC SKCOUNT     ;NO,DECREMENT COUNTER
0076: A9 01                     LDA #1          ;CLEAR ZERO FLAG(BUFFER NOT FULL)
PAGE -   3  GETTEXT   FILE:AFOREIGN

0078: 60                        RTS             ;THATS IT
0079:
0072* 00
0061* 7900
0079: A0 00             DOIT    LDY #0
007B: 91 04                     STA (BUFFER),Y  ;PUT CHAR IN BUFFER
007D: E6 04                     INC BUFFER
007F: D0**                      BNE SKIP1
0081: E6 05                     INC BUFFER+1    ;BUMP BUFFER POINTER
0083:
007F* 00
0083: E6 06             SKIP1   INC NBYTES
0085: D0**                      BNE SKIP2       ;BYTE SKIPPING
0087: E6 07                     INC NBYTES+1
0089:
0085* 00
0089: 38                SKIP2   SEC
008A: A5 02                     LDA MAXBYTES
008C: E9 01                     SBC #1
008E: 85 02                     STA MAXBYTES
```



A FORTH INTRODUCTION

J HOPPRICH

PERSONAL COMPUTERS LTD.
194-200 BISHOPSGATE, LONDON.

THE problem of choosing which programming language best fits a particular task is rarely faced by programmers since most of them tend to know only one or two languages, usually of similar type e.g. FORTRAN/BASIC or ALGOL/PASCAL. Even if they are familiar with the basic concepts of other languages they are not prefficient enough to construct complex programmes, causing familiar languages to be "bent" to fit the requirement at hand.

At present there are far too few programmers - let alone "multi-lingual" ones, so a remedy to the situation does not seem likely in the near future.

Many people, however, believe that a diverse range of language is unnecessary and should be cut down to a smaller number of more general purpose languages, and possibly to a single all-purpose language. This approach would lead to the creation of a large language that would, perhaps, be moderately good for most tasks but would fail to be excellent for any one task. An example of this would be PL1.

Language designers have always had the problem of deciding whether to create a large language suitable for most tasks but awkward to use, or a number of small specialised languages which are tediuos to learn and may only be used appropriately in one or two areas. The result is usually a compromise between these two extremes and most of the commonly used languages such as PASCAL or BASIC fall into this category; they are neither all embracing nor task specific.

FORTH, however, is a languge that differs fundamentally for the ones mentioned above. The FORTH package comes as a set of words in what is called the dictionary; each word performing some pre-defined task when executed. The execution may be in sequence to form a programme, but more importantly, this basic set of words allows the user to define words of his own and store these in the dictionary for future use.

By defining your own set of words you may steer the language in the direction you wish it to take, to fit the task in hand.

For example, a dictionary of words may be defined to operate the motor drive of a telescope and analyse the data from various instruments attached to it. Indeed this is what FORTH was first used for by its compiler Charles Moore in 1970 at the National Radio Astronomy Observatory in the U.S.A.

The basic words in the dictionary, mostly written in FORTH, are used for defining new words, manipulating the two stacks and writing or reading from memory locations.

The FORTH I use is version 1.5 for the APPLE, written by Bill Rogsdale and modified by John Draper. This version is not generally available in the U.K. but a new one, version 1.9 is in the pipeline from this team and may be released in this country soon. Another version by Programma International is available from Petsoft. The main difference is that the Programma version has disk commands written into the dictionary, (version 1.5 hasn't but version 1.9 will have), and this is also available for the PET (on tape only).

Loading FORTH brings the response:

APPLE FORTH V 1.5

and the system then awaits input. Typing in CAT followed by return executes this pre-defined word, in this case, printing out a catalogue of all the words with their positions in the dictionary. This will continue until the end of the dictionary is reached or a key is pressed to supress output. The system will then print:

OK

to indicate that it has no objection to what you have just done.

To create a word first type ":", which indicates a word is to be compiled for the dictionary. What immediately follows becomes the name given to the definition. All the subsequent items entered until ":", is used to indicate the end of the definition, are executed in order and must already exist in the dictionary.

For example, the input

: CATALOG CAT;

compiles the word CATALOG into the dictionary which when executed executes the word CAT described above. Note that each item must have a space before and after it including ":" and ";", as this feature is used

to mark the beginning and end of a single word. The definition name can consist of any ASCII characters except spaces and special control characters. Examples of word names are:
are:
VARIABLE ##2%IN 1## ?

You will notice that numbers can be used to name definitions, but this is best avoided as it caused confusion with actual numbers,

The word variable sets up a 16- bit memory location with a number and gives the location a name i.e.

35400 VARIABLE HI

assigns the word HI with the value 35400. Note that the number was entered before the word that used it i.e. 35400 VARIABLE and not VARIABLE 35400. The reason for this is that numbers entered are placed on top of the PARAMETER STACK which has a last-in-first-off operation. When variable is evoked it picks the top number off the stack and uses it as a parameter, leaving the stack as it was before the number was placed on top. Most of the dictionary words work in this way which is called Reverse Polish Notation and is also the method used by Hewlet Packard calculators. Because of this stack orientated system VARIABLE and CONSTANT, which associated a word with a constant number, are used only sparingly. There is also another stack - the RETURN STACK which handles the various system variables but can also be accessed by the user if need be.

FORTH also comes with a powerful screen editor and in the 1.5 version six, 1K byte screens. These are used to create a number of words in the way described above and edit them. There screens can be saved on tape or disc and loaded back into the screen areas when needed. The word LOAD, preceeded by the screen number, compiles a screen of words and places them on top of the dictionary. Typically 1K of source code will compile into approximately 200 bytes. Several sets of screens can be compiled into the dictionary in this way enabling it to be orientated towards several tasks without having all the words in the dictionary at the same time. This saves space, the basic system and dictionary occupies about 6K bytes and execution time as words are searched from the top of the dictionary.

Lets try to define a simple word and see what happens when it is evoked. The input would look like figure 1.

```
APPLE FORTH  V 1.5
: F->C 32 - 5 * 9 / . ; OK
212 F->C 100 OK
32  F->C 0 OK
100 F->C 37 OK
5000 F->C 2760 OK
MEM OK
3D  F->C 10 OK
FF  F->C 7B OK
ED  F->C 71 OK
MHO  BY MFR,  7/30/78 OK
FORTH OK
```

Fig. 1. Input for the Fahrenheit to Centigrade conversion.

The first line is the acknowledgement that the FORTH system is up and running. On the next line is the definition of a word:

F->C

which converts the number on top of the stack from Fahrenheit to Centigrade and prints the result. In the third line is an example of 212°F (input 212 F->C (return) being converted to 100°C. This is how the stack is affected: (See fig.2 for stack elelments and operations).

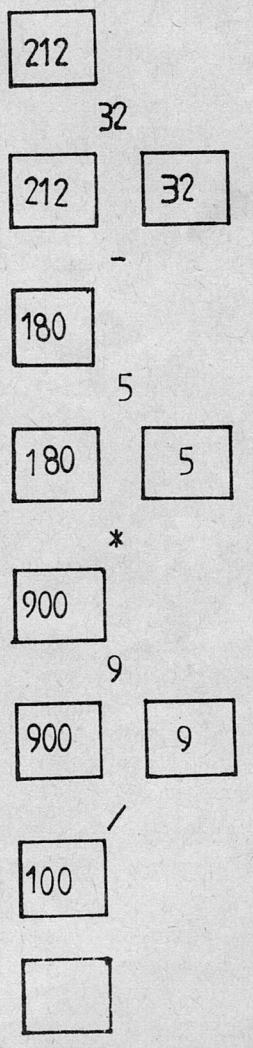


fig.2

212 is placed on top of the stack

Execution of F->C first of all places 32 on top of this so the stack now looks like:

212 32

Next these top two items are lifted from the stack, the uppermost number is subtracted from the second and the result if placed back on the stack:

180

The number 5 is then placed on top of this and "x" executed, multiplying these two numbers and leaving

the result on the stack as before:

900

The number 9 is placed on top of this and the word "/" is used to divide the uppermost item into the second again placing the result on top of the stack:

100

Finally, execution of the word "." takes the top item from the stack, prints it and leaves the stack empty, (both stacks are 16 bits wide).

It is interesting to note the effect of the word "HEX" at this point. All numbers are stored in binary and are converted to the present number base of the system which is held in a system variable. HEX changes the value of this location to 16 (base 10). Now all numbers are interpreted as hexadecimal, including the numbers placed on the stack from within a word definition, e.g. 32 (base 10) is now implemented as 20 (base 16). Any base can be used from 2 to 255, (the Upper limit), as long as you don't mind funny ASCII characters in your number set, such as " in base 36.

As a further note of interest the word "WHO" tells the author of this version and the date of its creation. Those of you who are following the "structured debate" (whether or not to have "structured" languages as "standard") will realise that as each word can only be defined by words already in the dictionary, FORTH leads necessarily to structured programming.

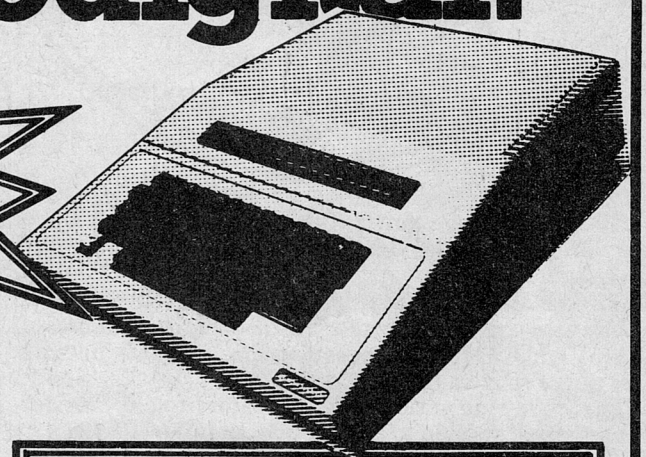
You may have noticed that in some ways FORTH is halfway between a high level language and an assembly/machine code language, this feature leads to Forth executing its commands extremely quickly, executing an empty loop on a tenth of the time of Integer Basic on the Apple. FORTH also has IF-THEN-ELSE, BEGIN-END and DO-LOOP structures making it a very powerful language.

Finally, I must express my deepest thanks to Gil Filbey for all his enthusiasm and the help he gave me when I was learning this language, he is currently writing a series of articles on FORTH in Computer Age. Also my thanks to Douglas Butter of FORTH INC. (U.K.) for much valuable information.

Isaac Newton's easy solution!

Apple prices are tumbling at Microdigital!

A 48K Disk based computer system for only **£1044** +VAT For March, April, May and June only.



	Nett	VAT	Total
16K byte Apple Computer	£695.00	£104.25	£799.25
Disk System	£349.00	£52.35	£401.35
Total with 32K free extra memory	£1044.00	£156.00	£1200.60

OFFICAL ORDERS

We Welcome official orders from bona-fide commercial and government organisations. We require payment 14 days after invoice date.

Latest Apple II plus model with floating point basic and auto-start ROM.

Buy a 16K Apple with Disk drive and get 32K of memory Free.

Apple Business Systems -ring Graham Jones 051-227 2535

Software Packages	£
Stock Control	£225.00
General Ledger	£295.00
Purchase Ledger	£295.00
Sales Ledger	£295.00

Word Processor/Letter Writer	£130.00
Book keeper Package	£295.00
Credit Control Package	£150.00
Payroll	£385.00

+ VAT at 15%

CRA

apple computer Sales and Service

Extras			
Second Disk Drive	£299.00	£44.85	£343.85
Pascal Language System	£299.00	£44.85	£343.85
Graphics Tablet	£462.00	£69.30	£531.30
Appletel System	£595.00	£89.25	£684.25
Black & White Modulator	£14.00	£2.10	£16.10
Eurocolour Card	£79.00	£11.85	£90.85
Joystick	£25.00	£3.75	£28.75

All Prices include courier delivery within U.K.

MICRODIGITAL

25 Brunswick Street, Liverpool L2 0BJ
Tel: 051-236 0707 (24 hour Mail Order)
051-227 2535 (All other Depts.)

Mail orders to: MICRODIGITAL LIMITED, FREEPOST (No Stamp Required) Liverpool L2 2AB

Please send systems as above at £1200.60 in addition please send:

Total Remittance _____

Name _____

Address _____

_____

Access and Barclaycard welcome.

Dr Andrew Veronis

President ACFA Inc.
P.O Box 1070 Glen Burnie
Maryland 21061
USA

AN INTRODUCTION

PART 2

CONTINUING this month with Pascal, I will introduce the various types of data which are handled by this language.

INTEGER DATA

Integers are represented by 16 bits for a range of -32767 « n « 32767. Expressions that evaluate outside this range cause a run time error.

The following arithmetic operators yield integer values when applied to integer operands:

\+ add
* multiply
\- subtract
DIV divide and truncate (not round)
MOD remaindering

The following relational operators yield BOOLEAN results when applied to integer operands:

= equal
«» unequal
« less than
«= less than or equal
»= greater than or equal
» greater than

The following standard functions yield integer values:

ABS(x) absolute value of x (x is the type integer)
SQR(x) square of x(x is the type integer)
TRUNC(x) truncates fractional part of type real x
ROUND(x) rounds the value of type real x

Conversions called implicit type conversions can be accomplished by assignment statements of the following general form:

integer variable: = expression yielding real number
real variable: = expression yielding integer number

REAL DATA

REAL data (numbers) are represented by 32-bit floating point numbers giving a range of 10 (-38) « n « 10(+38) with an appropriate amount of digit accuracy (usually 6).

An expression that evaluates to a value outside this range causes a runtime error.

The following arithmetic operators produce REAL values as long as at least one of the operands is of type REAL:

\+ add
* multiply
\- subtract
/ divide

Notice that real divides use the operator "/" and integer divides use "DIV". The following standard functions produce REAL values when called with REAL arguments:

ABS(x) absolute value of type REAL x
SQR(x) square of type REAL x
TRUNC(x) truncates fractional part of type REAL x
ROUND(x) rounds the value of type REAL x

The following standard functions produce real values when called with either REAL or INTEGER values:

SIN(x) sine
COS(x) cosine
ARCTAN(x) arctamgent
LN(x) natural log
EXP(x) exponential function
SQRT(x) Square root

and
'0' « '1' « '2' .....« '9'

Each character in the set has a corresponding integer value, called ordinal number, that is unique. The set of ordinal numbers ascend in the same fashion as the characters themselves, such that:

BOOLEAN DATA

BOOLEAN DATA are represented by either of the truth values TRUE, FALSE. The BOOLean operators are AND, OR, and NOT. The operator NOT is always applied first, then AND, and finally OR. This order can be altered by the use of parentheses. For example, if you made these declarations:

```
VAR
    DONE, NONE, TH, FLAG:  BOOLEAN;
```

then statements would yield Boolean values:

```
FLAG := DONE AND NONE OR TH;
FLAG := NOT NONE OR TH;
FLAG := TH AND (NONE OR DONE);
```

In addition to the Boolean operators, there are relational operators as follows:

```
«»  unequal
«   less than
«=  less than or equal
=   equal
»=  greater than or equal
»   greater than
```

Boolean operators and relational operators can be combined in expressions like these to produce a Boolean value of true or false:

```
(MIN « COUNT) AND (Y = X)
(SUM » MAX) OR EOF(INPUT) OR ERR
(INDEX = 20) AND NOT ERR
```

Finally, there are standard Boolean functions that return a value of TRUE or FALSE:

ODD(x) if x (must be integer) is odd, returns TRUE, otherwise FALSE.
EOLN(f) returns TRUE for end-of-line on textfile f; otherwise returns FALSE.
EOF(f) returns TRUE for end-of-file while reading file f; otherwise returns FALSE

The functions EOLN and EOF require a filename unless the filename is INPUT.

CHARACTER DATA

CHARACTER DATA consist of printable characters such as, alphabetic characters, numbers, and special characters including the blank character.
A CHAR literal is a single character enclosed in apostrophes, i.e., 'M', '2', '?'. To represent an apostrophe, you write it twice, and enclose the string in apostrophes, as in ''''. The set of alphanumeric characters is ordered such that the character values ascend according to the following:

'A' « 'B' « 'C' .....« 'Z'

Note that type REAL numbers and expressions cannot be used as arguments for the following:
1. Indexing arrays
2. Controlling FOR statements
3. Defining the base type of a set
4. Calling functions PRED and SUCC

ordinal ('A') « ordinal ('B') .....« ordinal ('Z')

The same applies to ordinal numbers.

There are two standard functions, ORD and CHR, that obtain the ordinal number for a given character or conversely obtain the character for a given ordinal number. This enables you to convert characters to numeric index values and to use the relational (coparison) operators to get BOOLEAN results.

The function ORD has argument of type CHAR and a result of type INTEGER. The function CHR has an argument of type INTEGER and a result of type CHAR. The identities are:

```
CHR(ORD(c)) = c
and
ORD(CHR(i)) = i
```

All the standard comparison operators «, «=, =, «», », »= can be used, and only these operators can be used on CHAR variables.

The relationship, r, of the ordered sets of characters and ordinal numbers is defined by:

```
C1 r C2
if and only if
ORD(C1) r ORD(C2)
```

There are two standard functions, PRED and SUCC, that can be used to obtain, respectively, the predecessor and successor or ordinal number. For example,

```
PRED(c) = CHR (ORD (c) - 1)
SUCC (c) = CHR (ORD (c) + 1)
```

TYPE DEFINITION PART

THE type definition part enables custom definition of data types beyond the standard integer, real, char, and boolean. For example,

```
TYPE
COLOR = (RED, YELLOW, BLUE, GREEN, WHITE);
DAY = (MON, TUESDAY, WED);
INDEX = 1..20;
WORKDAYS = MON...FRI;
TOOLKIT = SET OF TOOLS;
```

The data types can define fall into three general categories: scalars, structures, and pointers.

To define a new data type, you must provide an identifier (name) for it plus an ordered set of discrete values (by name), and it must be declared by a TYPE statement. The general form of the TYPE statement is:

```
TYPE type =type-definition;
     type =type-definition;
```

SCALAR TYPES

HERE are some possible user defined scalar types:

```
TYPE
RAINBOW = (VIOLET, BLUE, GREEN);
DAY = (MON, TUE, WED, THUR, FRI, SAT, SUN);
MOVE = (BACK, LEFT, RIGHT, FORWARD);
```

In the VARiable declaration section that follows the TYPE definition section, you then declare VARiables using these types. For example,

```
VAR
COLOR;RAINBOW;      (*COLOR is type RAINBOW*)
PAYDAY, HOLIDAY; DAY; (*PAYDAY and HOLIDAY
are type DAY*)
JUMP:MOVE; (*JUMP is type MOVE*)
```

The items in the list are ordered in ascending value from left to right, such that:

```
VIOLET « BLUE « GREEN ... « RED
MON « TUE « ... « SUN
                «
BACK « LEFT « RIGHT « FORWARD
```

The function ORD provides the ordinal number of an item within a list of scalars, where the first item in the list has ordinal number zero, the second item has ordinal number 1, the nth item has ordinal n-1. Example:

```
ORD(VIOLET) = 0
ORD(BLUE) = 1
ORD(WED) = 2
ORD(FORWARD) = 3
```

The functions PRED and SUCC can be used to determine the predecessor or successor of a scalar (except REAL). For example, using RAINBOW, DAY and MOVE above:

```
SUCC(BLUE) = GREEN
SUCC(VIOLET) = BLUE
PRED(WED) = TUE
PRED(FORWARD) = RIGHT
```

A given value cannot belong to more than one type. For example, the following is illegal because of a mixed assignment or ORANGE:

```
TYPE
RAINBOW = (VIOLET, BLUE, GREEN, YELLOW,
ORANGE, RED);
FRUIT = (ORANGE, LEMON, APPLE);
```

Valid statements are:

```
COLOR :=YELLOW;
PAYDAY :=FRIDAY;
```

Mixed assignments are not allowed; for example, you could not write:

```
JUMP :=BLUE;
```

SUBRANGE TYPES

A subrange is a proper subset of a previously defined type scalar (except REAL). A subrange is specified by two constants: the lowerbound and the upperbound. The example below shows COLORSUBRNG, WORKDAYS, and WEEKEND as subranges of types RAINBOW and DAY:

```
TYPE
RAINBOW = (VIOLET,BLUE,GREEN, YELLOW,
ORANGE, RED);
DAY = (MON,TUE,WED,THUR,FRI,SAT,SUN);
COLORSUBRNG = GREEN ... RED;
WORKDAYS = MON ... THUR;
WEEKEND = FRI..SUN;
```

The general form for a subrange is:

```
TYPE type = constant..constant;
     type = constant..constant;
     .
     .
     .
```

Subranges are particularly useful in setting up range checking, because any variables that go outside specified ranges cause a runtime error. In the subrange sets (I..J), the valid values for I and J are I» = 0 and J« = 16*255.

Subrange variables are declared in the same way as for type INTEGER and CHAR. Fro example,

```
VAR
WIRECOLOR: COLORSUBRNG;
XYZ: INTEGER;
PAYDAY: WORKDAY;
OTDAY: WEEKEND;
```

Since the types INTEGER and CHAR are already defined, you can also use subranges like:

```
TYPE
PREFIX = 'A'..'K';
LIMIT = 1..25;
```

SET TYPES

Set types are defined as a group of items of the same type, and sets must contain scalars only. The examples given in this document are based on the following TYPE statement:

```
TYPE
TOOLS = (HAMMER, SAW, LEVEL, SCREW-
DRIVER, SANDER);
```

From the discussion on types, you know that the above type statement declared a list of values that can be assumed by a variable of type TOOLS, and that the items in this list are an ordered sequence. To declare an actual set, for example,

```
TYPE
TOOLBOX = SET OF TOOLS;
```

This statement sets up the type TOOLBOX as a set of type TOOLS that can contain none or any or the items listed. You can now declare VARiables of type TOOLBOX, and these variables are sets that can be operated on by set operators discussed. Here are possible VARiable declarations:

```
VAR
WOODTOOLS, PIPETOOLS, METALTOOLS : TOOL-
BOX;
BIGJOB, BOX1, SMALLJOB: TOOLBOX
```

Set members are enclosed in square brackets as in the following assignment statements:

```
WOODTOOLS : = (HAMMER, SAW, SANDER):
PIPETOOLS : = (LEVEL, PIPEWRENCH);
BOX1 : = WOODTOOLS + (PIPEWRENCH);
```

Consequently, set members can be specified by first and last members only, such as:

```
BOX1 : = (HAMMER..SANDER) ;
```

An empty set is specified by a statement like:

```
BOX1 : = ();
```

The set union operator (similar to logical AND) is specified by '+'. The resultant set contains all members of all sets. For example, the statement

```
BIGJOB : =WOODTOOLS + PIPETOOLS + METAL-
TOOLS;
```

gives you the set BIGJOB containing all the tools of all three sets. Note that common items such as SANDER are only named once.

The set intersection operator (similar to logical OR) is specified by '*'. The resultant set contains only those items that are members of all sets specified. The statement:

```
SMALL JOB : =WOODTOOLS * PIPETOOLS *
METALTOOLS;
```

would yield the set SMALLJOB, containing only SANDER.

The set difference operator is '_'. The resultant set SMALLJOB contains all members of the first set that are not members of the second set.

The relational operators can be used to compare sets:

Operator	Example
= set equality	IF BIGJOB = HAMMER..BRAKE.....
etc	

The reserved word IN is used to test set membership:

```
IF METALSHEARS IN BOX1 THEN ...
IF NOT ((VISE OR SCRIBE)) IN BIGJOB THEN..
```

NOTE: A set can be at most 255 words in size, and have at most 4080 elements.

Comparisions and operations on sets are allowed only between sets which are either of the same base type or subranges of the same underlying type. For example, in the sample program below, the base type of the set S is the subrange type 0..49, while the base type of the set R is the subrange type 1..100. However, the underlying type of both sets is the type INTEGER, which (by the above definition of compatibility) implies that the comparisions and operations on the sets S and R in the following program are legal:

```
PROGRAM SETCOMPARE;

VAR
S: SET OF 0..49;
R: SET OF 1..100;

BEGIN
S: = 0,5,10,20,25,30,35,40,45   ;
R: = 10,20,30,40,50,60,70,80,90   ;
IF S = R
   THEN WRITELN ('...oops...')
   ELSE WRITELN ('sets work');
S : = S+R
END.
```

However, in the following example, the construct is not legal since the two sets are sets of two distinct underlying types:

```
PROGRAM ILLEGALSETS;

TYPE STUFF = (ZERO,ONE,TWO);
VAR
   I: SET OF STUFF;
   J: SET OF 0..2;
```

PACKED VARIABLES

A pascal compiler performs packing of arrays and records if the ARRAY or RECORD declaration is preceded by the word PACKED.

No element of a PACKED ARRAY to field of a PACKED RECORD can be passed as a variable (call--by-reference) parameter to a PROCEDURE or FUNCTION. Packed variables can, however, be passed as call-by-value parameters.

Next month we will continue our discussion with Arrays and other interesting Pascal topics. See you then.

1st GRADE EPRONS

Through our buying we can offer the following:

2716-3 RAIL - £20.00
2716-Single 5V-£20.00

including VAT and Postage

COMPETITIVE VIDEO
ELECTRONIC GAMES
BERESFORD COURT, BERESFORD ROAD,
NEW MALDEN SY.
Tel: 01-949-3327

A Beginners Introduction to Pascal

T.A.Dodd

YOU'VE read about the system introduced by Apple and realised that Pascal is a powerful beginners language. But can an enthusiast get by with one disk drive, the Pascal Language Kit and little knowledge? Being very satisfied with my fortnight old system I hope here to describe (in simple terms) the difficulties of the first time user, give an inkling of the excellent facilities available on the Apple II Europlus Pascal system, and comment on the advantages of Pascal for the novice not the professional.

The Basic Language Autostart ROM allows easy start up straight into Applesoft. Installing the disk drive is extremely simple, slot 6 being the preferred position. Putting an ITT 2020 Dual System Master Diskette version 5.3 (Dos. 3.2) into the drive, then switching MAINS 'ON' gives the Basic of your choice by keying FP return for Applesoft I or INT return for Integer », the cursor indicating which Basic is in use. The mini-assembler is available if the monitor is entered from Integer Basic. (»CALL-151, *F666G). This is routine stuff. What about Pascal? Dallying in Basic just long enough to check that the hardware functions, the Language Card kit is acquired. "Just like that", did somebody say?

Installing the Language card is straight forward. Two PROMS are removed from the disk controller card and two new ones fitted. A RAM chip is removed from the board and the Language Card ribbon cable plugged into the vocated socket. The Language card plugs into [illegible] 0.

If you think PROM removals are best left to the professionals but can't afford them, take heart. Full instructions are provided and should be read. Having completed the job on my rig I later reversed the process, re-fitting the original PROMS and RAM. Even second time round there were no mis-haps. This lapous anima was caused by ignoring the rule "if all else fails read the instruction". However, there are a lot of instructions. I mention this to show that changing the PROMS requires care but not anxiety. The whole job takes less than ten minutes.

It is essential at this stage to have two spare diskettes. One is available with the purchase of the disk drive, while a spare is usually included with the Language card. The object is to copy Pascal diskettes APPLE0: and APPLE3:. Only when the master diskettes have been copied will the system run. This also applies to the twin-disk system which uses APPLE1: and APPLE2: (it says here).

Having initialised the first diskette in Applesoft, a mistake at that time it turned out, I formatted the second for Pascal using the easy to follow detailed instructions on page 289 of the Pascal Manual. Formatting was easy. With APPLE3: in the disk drive just switch MAINS 'ON', put APPLE0: in the drive and hit reset. This becomes a set routine for powering up since the two Basics and monitor can be reached quickly from Pascal. Here Pascal gives a welcome message and the screen shows you are in the command mode with several options available next. Since this signals the first contact with a Pascal implementation I would like to mention the comprehensive article 'Apple Pascal' by Dr. Martin Beer of 'Computer Age', issue 2, January 1980 and 'Pascal Bytes the Apple' by C. Phillips in 'Liverpool Software Gazette', November 1979.

The actual Pascal formatting method is to put APPLE3: in the drive and select X. When asked what file is to be executed, type in APPLE3: FORMATTER and hit RETURN. Put in your new diskette and hit 4 and return. "Now formatting diskette in drive 4" appears, and when that is done "Format which disk (4, 5, 9 ... 12)?" appears. 4 means slot 6, drive 1. Remove your newly formatted disk and press return. You are told to put System Disk in 4 and press return, so insert APPL0: in the drive and return. 'That's all folks' it says and you are back in command mode.

When the message 'Format which disk' appears you can easily substitute another new disk and press return. As formatted, the disks are called BLANK: and will be so called later. Return to command level is as above i.e.

remove disk BLANK: (as it is now), press return, insert APPLE0: and press return. All diskettes require formatting.

To make a copy, starting from command mode with APPLE0: in drive you type F for Filer then T for transfer. Reply to the screen by typing in say APPLE0: return, (the : indicates the whole diskette). 'To where' is answered by typing BLANK: return. Then follow the instructions on the screen for four minutes. This involves substituting one disk for another. At the end you are returned to the Filer, so type D for date then type in the day's date. This will re-appear in the SYSTEM.WRK file when you create and save or update a work file. Q for quit returns you to the command mode. On the inside rear cover of the Pascal manual is a tree of the options available, and they are given in both the quoted articles, a useful reference.

My mistake of initialising my only other available diskette for BASIC led to some problems formatting it for Pascal but in fact the basic was ignored and I ended up with the required read/write copies. Remembering there were 287 introductory pages to cover, I realised that the confronting 'Press this' 'look at that' nature of the paperwork had disappeared. However, I had discovered that Apple will work with any floppy whether it is 10 sectored or soft sectored, and later I made a good copy on a multisectored high quality disk in case my 'converted from basic' disk ever gave trouble. It hasn't yet.

The story so far is that I've installed the Pascal interface, made read/write copies of Disks APPLE0: and APPLE3: and want to write something resembling Pascal, compile it, and see it run. This is where the excellently constructed machine showed the quality of its software. It's the software that converts this heap of primordal sand and copper into a powerful machine. To introduce some brevity I'll take a strikingly trivial example to illustrate the machine's forgiving nature, and try not to become hysterical about the nearly 400 message diagnostics which almost allow you to learn syntax from the machine. However, how to enter the programme?

Powering up with APPLE3: in drive, you insert APPLO: and press reset to enter Pascal in the outer Command level.. To go to Basic insert BASICS:, press H, and get the message "Insert Basic Disk and press return key". Insert a disk initialised in either Basic and press return and you are in that Basic. This is why its a good idea to have a Basic initialised disk. Return to Pascal by booting APPLE3: with PR 6 or C6OOG, insert APPLE0: and reset. You may notice the disk whirr pause, then whirr again before the Pascal Welcome appears for the second time since power up. Changing modes is that easy, but we still want to enter a trivial Pascal programme to check it works, so we can move on to use those 36 number long integers, the eight character variables, the formatted WRITE statements the unit to give a note from the built in speaker, or whatever.

The method is to start in the command mode type E for Edit leaving APPLE0: in the drive at present. In Edit mode you may be told no file is present, but any file there may be removed by typing F//N//Q. F puts you in Filer, N for New clears the workfile and Q quits the Filer and back to the Command mode. Go back to the Filer and try E for Extended Directory. The message "DIR listing of?" prompts the reply APPLE0: (the disk in drive). Pressing return gives a list of all files and unused files. Notice there is no SYSTEM.WRK. Text or code. The program is text and the compiler generates the code. The top of the screen still lists the Filer options and keying ? causes those not displayed to be displayed. Since Quit is available we press Q to go back to the Command code.

E puts us in the Edit mode with no workfile present. The options run off the screen so press CTRL-A to see the RHS 4D cols. Hold down CTRL and press A to return to the LHS page. Having over ensured in three minutes or less that there is no workfile present (and it says so on the screen) we press return and are still in the Edit mode ready to insert a programme and select CTRL-C to have the system accept it. Next time you visit the Editor you can shove the program all over the place with apparently consumate ease.

This is where the star studded software really is a joy to use. In EDIT mode first hit I. The program is typed in just as you want to see it, then pressing CTRL-C (hold down CTRL and hit C) causes the system to accept the insertion. To Quit the editor press Q and you are offered the choice of Up-date (save on cassette), Exit without update, Return to Editor without update of work file, or Write to a File Name and return. Suppose we select U for update. We will compile our program and selecting R will first compile an uncompiled program then run it or if the workfile is already compiled it just runs it.

That is an overview of the method and again can be followed using either of the quoted references. To use a mighty sledge hammer to crack a tiny nut the delay timer or code timer fragment given in Basic by:

```
1 PRINT "START"
2 FOR X = 1 TO 5000
10 NEXT X
11 PRINT "END"
12 END
```

can be used. The time between START and END appearing on the screen gives the time for 5000 FOR ... NEXT loops to be executed. In most basics this is around 10 secs. Segments of code can be inserted between 2 and 10 so that the extra time measured on a watch is their time of execution for 5000 operations. Some basics don't allow 5000 loops in a FOR ... NEXT statement.

To enter Pascal programme simply follow the syntax for a complete programme, a block and statement. To play a few notes as audible warning requires an APPLESTUFF unit procedure called NOTE which as as parameters variables called PITCH and DURATION. Since the speaker routine is contained in a system file called APPLESTUFF we tell the Pascal system to get the NOTE procedures from the Library by the phrase USES APPLESTUFF; A path to include this option is shown in the Pascal manual but not in all other books. Another easy feature is that the (identifier) part of the programme syntax diagram may be omitted. The identifier for the program can be NONSUCH, and in the block we can skip down past LABEL (there's no requirement to exit to a predifind point in the middle of a loop), past CONSTant (no need for PI = 3.14159;) past TYPE (no need to type say COLOUR or DAYS OF THE WEEK) and just declare VARiables PITCH, DURATION, etc., as of type INTEGER, and go out via:

which can lead down to - BEGIN statement END, back to the programme diagramme to finish off with, which leaves us to fill in the statement part. Any errors in typing or syntax will be caught in the compiler. Since the first eight characters in a variable are recognised you could use a lot of variables and it helps to spell them correctly when you declare the. In crew room bridge contests, if you shouted six hearts when you meant six diamonds you could expect your growth to become stunted. Sim-sim in computers, one supposes.

```
PROGRAM NONSUCH;
USES APPLESTUFF;

VAR PITCH, DURATION, COUNT, TOTAL :
INTEGER;

BEGIN
  WRITELN ('HERE');
      FOR COUNT: = 1 TO 5000 DO
        BEGIN
          TOTAL: = 3*5
        END;
WRITELN ('THERE' :9)
DURATION: = 77;
FOR PITCH: = 12 to 24 DO NOTE (PITCH, DURA-
                                    TION);
END.
```

The use of BEGIN ... END may seem cumbersome, but seen on the screen it can mark off the space in which you wish to insert some code. Widening the gaps between the BEGIN at the top and the END lower down just go into EDIT mode (eg type E in command mode) holding down the key marked CTRL and press O for up and L for down to get the correct line, hit I for Insert then step the cursor down with as many RETURNS as are required. Hit CTRL-C to have the correction accepted and the program is written out again.

Why have I gone in for all this 'press A' 'look at B' stuff when many readers are interested in technical data? Mainly because many people can visualise the actions or may try them given the opportunity. And the bare statement that certain modes are available can't convey the delight of operating ones way around all the options with few mistakes as most enthusiasts will enjoy doing. Step over cursors are common in a lot of machine code routines right now, and the insert and delete facility might look good on other Text Editors as a matter of course.

To comment on the program, the UCSD WRITE facility can be modified by adding a number :9 which allocates 9 spaces into which the stuff to be written is inserted. In the examples, the T in THERE lines up with the E in HERE because the addition of LN to a WRITE statement causes a Line Feed/Carriage Return, and then the :9 allows 9 spaces into which ... THERE is written. Do away with creative guessing of the number of blanks to line up text and numbers.

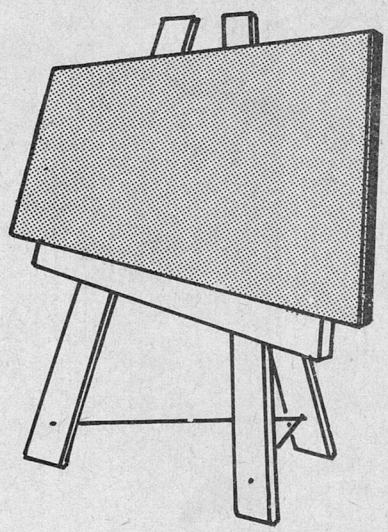
Incidentally, since the example was left in Edit mode, positioning the cursor (CTRL-O, CTRL-L), right arrow, left arrow are used) at the start of a line, selecting A for adjust allows the whizz kid to press right arrow key to move the whole line to the right. How to get out of that mode? Press CTRL-C to have the correction accepted and its all back in Edit mode. Similarly for left move.

Quitting the edit mode we can save the workfile by use of the Update command and then Compile the 'programme'. Running it we find the time for just 5000 loops is around 2 secs. In most Basics this would take about 10 secs. You could insert segments of code between the BEGIN ... END where TOTA = 3*5 is placed and measure the time they take for 5000 repititions. The numbers in Pascal are of course written without commas. If you want to write a , in a message just 3/8ut it in twice e.g. WRITE ('LET"S') gives LET'S at write time.

The USES APPLESTUFF; part of the program is similar to the USES TURTLEGRAPHICS, APPLESTUFF; used to furnish other units to the user. The integers are type LONG (pure jargon) but a minimum space can be allocated to any integers by using :INTEGER (22); instead of just :INTEGER; when 22 spaces will always be allocated for that integer as a minimum. Square brackets are not marked but are CTRL-K and SHIFT-M.

While I wouldn't advise anyone to buy into Pascal to keep up with the J.B. Page's (if only I could write like him) anyone thinking of getting a Language Card could have a present glimpse of an exiting future. The wealth of structures and data types, and the possibility of readable programmes and your own library of favourite procedures and functions and the built in facilities will keep any enthusiast quite happy for a very long time. The CASE ... OF construct is the novices big example of the difference between 'book Pascal' and UCSD Pascal, in that if the case is not specified UCSD fall through whereas books sometimes say that if a matching case is not found the result is indeterminate and also use the CASE ... OF statement with integers, which doesn't work with UCSD. However, there are so many facets to Pascal which seem to me to be useful in aiding all the methods which could make easier the transformation of algorithms into programmes which work, that it could be worth anybody looking further into it. Doodling on the screen with Pascal statements easily moved by the Editor could be fun.

A USEFUL PASCAL PROGRAM !



PHIL NOTT

IT is often useful when embarking upon setting up an APPLE UCSD PASCAL SYSTEM to transfer existing programs or data files from mini or mainframe systems.

However users will find that having used the intrinsic transfer routines to transfer text the editor will not accept it. The reason for this is that most systems use particular control characters to mark such things as beginning and end of filesor lines.

These characters unfortunately confuse the System Editor. The following program successfully removes all unwanted control characters from the incoming text and stores it in a 16K byte buffer which is then written to the disk. The editor will now accept the text.

This routine is machine-independant as regards the source of incoming text and may be adapted to operate on any UCSD Pascal system by altering the Card & System equates in the assembly routine GETTEXT.

This program can save considerable time in retyping, this is particularly useful to such a bad typist as myself!!!

```
 1   1    1:D     1  (*$LPRINTER:*)
 2   1    1:D     1  (*$P *)
 3   1    1:D     1 PROGRAM FOREIGN;
 4   1    1:D     3
 5   1    1:D     3 (*************************************************)
 6   1    1:D     3 (*  ACCEPTS TEXT VIA COM CARD USING ASSEMBLY     *)
 7   1    1:D     3 (*                                                *)
 8   1    1:D     3 (*  ROUTINE GETTEXT WHICH STRIPS OF UNWANTED CHARS *)
 9   1    1:D     3 (*                                                *)
10   1    1:D     3 (*    IMPLEMENTATION : PHIL NOTT                  *)
11   1    1:D     3 (*                                                *)
12   1    1:D     3 (*    SYSTEM/LANGUAGE: APPLE II+ , UCSD PASCAL    *)
13   1    1:D     3 (*                                                *)
14   1    1:D     3 (*    DATE           : FEB 1980                   *)
15   1    1:D     3 (*                                                *)
16   1    1:D     3 (*    THE AUTHOR OF THIS IMPLEMENTATION HEREBY    *)
17   1    1:D     3 (*    ACCEPTS NO RESPONSIBILITY FOR ANY DAMAGE    *)
18   1    1:D     3 (*    CAUSED BY ITS USE.                          *)
19   1    1:D     3 (*                                                *)
20   1    1:D     3 (*************************************************)
21   1    1:D     3  CONST BUFLEN=16000;
22   1    1:D     3       LINELEN=132;
23   1    1:D     3
24   1    1:D     3  VAR    BUFFER                                : PACKED ARRAY[0..BUFLEN]OF CHAR;
25   1    1:D  8004    NLINES,I,NBYTES,SKIPBUFS,SKIPBYTES,BUFPTR : INTEGER;
26   1    1:D  8010         S                                    : STRING[255];
27   1    1:D  8138         FNAME                                :STRING[30];
28   1    1:D  8154         F                                    : TEXT;
29   1    1:D  8455
30   1    2:D     3  FUNCTION GETTEXT(VAR BUFFER;MAXBYTES,SKIPBYTES:INTEGER):INTEGER;
31   1    2:D     6  EXTERNAL;
32   1    2:D     6
33   1    3:D     1  PROCEDURE GOODBYE;
34   1    3:D     1
35   1    3:0     0  BEGIN
36   1    3:1     0    WRITELN;WRITELN('BYE-BYE KIDDIES');
37   1    3:1    43    EXIT(PROGRAM);
38   1    3:0    47  END;
39   1    3:0    60
40   1    4:D     3  FUNCTION GETSTRING:BOOLEAN;
```

```
41  1   4:D    3
42  1   4:0    0  BEGIN
43  1   4:1    0    IF BUFPTR >=NBYTES THEN GETSTRING:=FALSE
44  1   4:1    9    ELSE
45  1   4:2   14      BEGIN
46  1   4:3   14        I:=SCAN(NBYTES-BUFPTR,=CHR(13),BUFFER[BUFPTR]);
47  1   4:3   39        GETSTRING:=TRUE;
48  1   4:3   42        IF I> LINELEN THEN
49  1   4:4   51          BEGIN
50  1   4:4   51          (*$R-*)
51  1   4:5   51           SC0]:=CHR(LINELEN);
52  1   4:5   59           MOVELEFT(BUFFER[BUFPTR],SC1],LINELEN);
53  1   4:5   73           (*$R+*)
54  1   4:5   73           BUFPTR:=BUFPTR+LINELEN;
55  1   4:5   83           I:=I-LINELEN;
56  1   4:4   93          END
57  1   4:3   93        ELSE
58  1   4:4   95          BEGIN
59  1   4:4   95          (*$R-*)
60  1   4:5   95           SC0]:=CHR(I);
61  1   4:5  103           MOVELEFT(BUFFER[BUFPTR],SC1],I);
62  1   4:5  117           (*$R+*)
63  1   4:5  117           BUFPTR:=BUFPTR+1+1;
64  1   4:4  129          END;
65  1   4:2  129      END;
66  1   4:0  129 END;
67  1   4:0  142
68  1   5:D    1 PROCEDURE INTRO;
69  1   5:D    1
70  1   5:0    0 BEGIN
71  1   5:1    0 WRITELN('ACCEPTS TEXT VIA COM CARD USING ASSEMBLY');
72  1   5:1   60 WRITELN('----------------------------------------');
73  1   5:1  120 WRITELN('ROUTINE GETTEXT WHICH STRIPS OF UNWANTED');
74  1   5:1  180 WRITELN('----------------------------------------');
75  1   5:1  240 WRITELN('CHARACTERS (CARD IN SLOT 2 ONLY)');
76  1   5:1  292 WRITELN('----------------------------------------');
77  1   5:1  352 WRITELN;
78  1   5:1  360 WRITELN('    IMPLEMENTATION : PHIL NUTT');
79  1   5:1  410 WRITELN;
80  1   5:1  418 WRITELN('    DATE           : FEB 1980');
81  1   5:1  467 WRITELN;
82  1   5:1  475 WRITELN('========================================');
83  1   5:1  536 WRITELN('  THIS ALLOWS BYTES TO BE SKIPPED AT THE BEGINNING OF EACH LINE');
84  1   5:1  619 WRITELN('  OF INCOMING TEXT AND PROVIDES THE CAPACITY FOR LARGE PIECES');
85  1   5:1  700 WRITELN('  OF TEXT OR PROGRAMS TO BE DIVIDED INTO 16K BYTE LENGTHS.');
86  1   5:1  778 WRITELN('N.B. 16 K IS A REASONABLE SIZE FOR THE EDITOR TO COPE WITH.');
87  1   5:1  857 WRITELN('  THE ROUTINE IS USEFUL IN THAT IT DOES NOT ACCEPT ANY ASCII');
88  1   5:1  937 WRITELN(' CHARACTERS THAT WILL UPSET THE EDITOR AND HENCE CAN ACCEPT TEXT');
89  1   5:1 1021 WRITELN(' FROM ANY MACHINE AND PROCESS IT WITHOUT ERROR.');
90  1   5:0 1088 END;
91  1   5:0 1100
92  1   1:0    0 BEGIN(* MAIN *)
93  1   1:1    0 INTRO;
94  1   1:1   17 REPEAT
95  1   1:1   17     (*$I-*)
96  1   1:2   17     REPEAT
97  1   1:3   17       WRITELN:WRITELN;
98  1   1:3   29       WRITE('OUTPUT FILE NAME ?');READLN(FNAME);
99  1   1:3   73       IF FNAME='' THEN GOODBYE;
100 1   1:3   85       IF POS('.TEXT',FNAME)=0 THEN FNAME:=CONCAT(FNAME,'.TEXT');
101 1   1:3  142       REWRITE(F,FNAME);
102 1   1:2  153       UNTIL IORESULT=0;
103 1   1:2  159       (*$I+*)
104 1   1:2  159       WRITELN('SKIP HOW MANY BYTES OF EACH LINE ?');
105 1   1:2  213       WRITE('(22 FOR TEDOS LISTINGS)');READLN(SKIPBYTES);
106 1   1:2  267       WRITE('SKIP HOW MANY BUFFERS ?');READLN(SKIPBUFS);
107 1   1:2  321       NLINES:=0;
108 1   1:2  325       WRITELN('FIRE WHEN READY....');
109 1   1:2  364       WRITELN('(<CR> WHEN TRANSFER COMPLETE)');
110 1   1:2  413
111 1   1:2  413 (*************************************************************************)
112 1   1:2  413 (*                                                                       *)
113 1   1:2  413 (* USE GETTEXT TO SKIP OVER THE FIRST SKIPBUFS BUFFERS                   *)
114 1   1:2  413 (* THIS LETS YOU BREAK UP BIG FILES INTO SMALLER ONES BY SENDING         *)
115 1   1:2  413 (* SEVERAL TIMES AND CAPTURING DIFFERENT PORTIONS OF THE TEXT           *)
116 1   1:2  413 (*                                                                       *)
117 1   1:2  413 (*************************************************************************)
118 1   1:2  413
119 1   1:2  413       FOR I:=1 TO SKIPBUFS DO NBYTES:=GETTEXT(BUFFER,BUFLEN-2000,SKIPBYTES);
120 1   1:2  461
121 1   1:2  461       NBYTES:=GETTEXT(BUFFER,BUFLEN,SKIPBYTES);
122 1   1:2  476       WRITELN(NBYTES,' BYTES RECEIVED');
123 1   1:2  523       BUFPTR:=0;
```

```
124   1    1:2   527
125   1    1:2   527       WHILE GETSTRING DO
126   1    1:3   533         BEGIN
127   1    1:4   533           WRITELN(F,S);
128   1    1:4   553           NLINES:=NLINES+1;
129   1    1:3   561         END;
130   1    1:3   563
131   1    1:2   563       CLOSE(F,LOCK);
132   1    1:2   572       WRITELN(NLINES,' LINES WRITTEN TO ',FNAME);
133   1    1:1   634     UNTIL FALSE;
134   1    1:0   637 END.
PAGE -   0
Current memory available:   10142
0000¦
0000¦                               .MACRO POP
0000¦                               PLA
0000¦                               STA %1
0000¦                               PLA
0000¦                               STA %1+1
0000¦                               .ENDM
0000¦
0000¦                               .MACRO PUSH
0000¦                               LDA %1+1
0000¦                               PHA
0000¦                               LDA %1
0000¦                               PHA
0000¦                               .ENDM
0000¦
0000¦
2 blocks for procedure code  9331 words left
PAGE -   1  GETTEXT   FILE:AFOREIGN


0000¦                                   .FUNC GETTEXT,3
Current memory available:    9553
0000¦
0000¦                           ;-------------------------------------------------
0000¦                           ;
0000¦                           ;  FUNCTION GETTEXT(VAR BUFFER;MAXBYTES,SKIPBYTES:INTEGER):INTEGER;
0000¦                           ;
0000¦                           ;  RETURNS # OF BYTES READ FROM COM CARD SLOT 2 BEFORE <CR> PRESSED
0000¦                           ;
0000¦                           ;  ON APPLE.
0000¦
0000¦                           ; ZERO PAGE EQUATES
0000¦                           ; -----------------
0000¦ 0000                      RETURN  .EQU    0
0000¦ 0002                      MAXBYTES .EQU   2
0000¦ 0004                      BUFFER  .EQU    4
0000¦ 0006                      NBYTES  .EQU    6
0000¦ 0008                      CHARS   .EQU    8
0000¦ 000A                      SKBYTES .EQU   0A
0000¦ 000C                      SKCOUNT .EQU   0C
0000¦
0000¦                           ; CARD & SYSTEM EQUATES
0000¦                           ; ---------------------
0000¦ C0AE                      STATUS  .EQU   0C0AE    ;DATA STATUS REG COM CARD
0000¦ C0AF                      DATA    .EQU   0C0AF    ;INCOMING DATA
0000¦ C000                      KEYBOARD .EQU  0C000    ;KEYBOARD BUFFER
0000¦ C010                      KEYRESET .EQU  0C010    ;
0000¦
0000¦                           ; OPERATING CODE
0000¦                           ; --------------
0000¦
0000¦                                   POP RETURN      ;SAVE RETURN ADDRESS
0000¦ 68                        #       PLA
0001¦ 85 00                     #       STA RETURN
0003¦ 68                        #       PLA
0004¦ 85 01                     #       STA RETURN+1
0006¦ 68                                PLA
0007¦ 68                                PLA
0008¦ 68                                PLA             ;4 BYTE STACK BIAS
0009¦ 68                                PLA
000A¦
000A¦                                   POP SKBYTES     ;HOW MANY BYTES TO SKIP AFTER <CR>.
000A¦ 68                        #       PLA
000B¦ 85 0A                     #       STA SKBYTES
000D¦ 68                        #       PLA
000E¦ 85 0B                     #       STA SKBYTES+1
0010¦                                   POP MAXBYTES    ;MAX. BUFFER LENGTH
0010¦ 68                        #       PLA
0011¦ 85 02                     #       STA MAXBYTES
0013¦ 68                        #       PLA
0014¦ 85 03                     #       STA MAXBYTES+1
```

ROL	axe	Temp - C;C -M(7);M(q) M(q-1);M(0) - Temp	CNZ; V=0 if C=N
ROR	axe	Temp - C;C - M(0);M(p-1)- M(7)-Temp	CNZ;
RTI		Regs 1	V=0 if C=N
RTS		P1	-
SBA		A - A-B	NVZC
SBC	a mdxe	a - a-M - C	nvzc
SEC		C - 1	C=1
SEI		I - 1	I=1
SEV		V - 1	V=1
STA	a dxe	M - a	NZ;V=0
STS	dxe	M - S	
STX	dxe	M - X	NZ;V=0
SUB	a mdxe	a - a -MM	NVZC
SWI		Regs 1; PC-interrupt rector;I-0	I=0
TAB		B-A	NZ;V=0
TAP		CC - A	HINVZC
TBA		A - B	NZ;V=0
TPA		A - CC	
TST	axe	M - 0	NZ;C=V=0
TSX		X - S + 1	
WAI			

Operand Key

one operand if all options close spaced, e.g.: axe
two operands if space between options, e.g.: a dxe

a = accumulator	= A or B
d = direct	EA = 1 byte address for page 1
e = extended	EA = 2 byte absolute address
m = immediate	EA = 1 byte actual data
r = relative	EA = (P + 2) = 1 byte displacement
x = indexed	EA = X ± 1 byte displacement

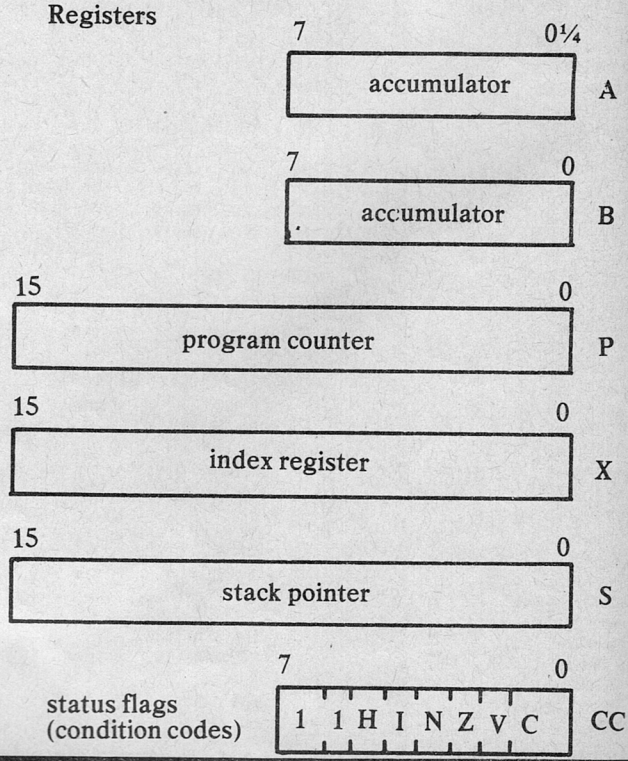
C = carry
H = half carry
I = interrupt
N = negative
V = overflow
Z = zero

= overflow to 'ninth bit'
= overflow from bit S
= maskable interrupt disabled
= bit 7 or operand
= two's complement overflow
= all bits zero

Operation Key

EA	= effective address
M	= memory location, data or accumulator referenced
M(n)	= nth bit of M
M	= complement of M
p	= from 1 to 7
from 7 to 1	
Regs	= all registers
(S)	= location pointed to by S
Temp	= temporary store
	= logical and
v	= logical inclusive or
o	= logical exclusive or
+	= plus
-	= minus
=	= is equal to
=	is not equal to
	= is replaced by
	= pushed back onto stack
	= popped off stack

Registers

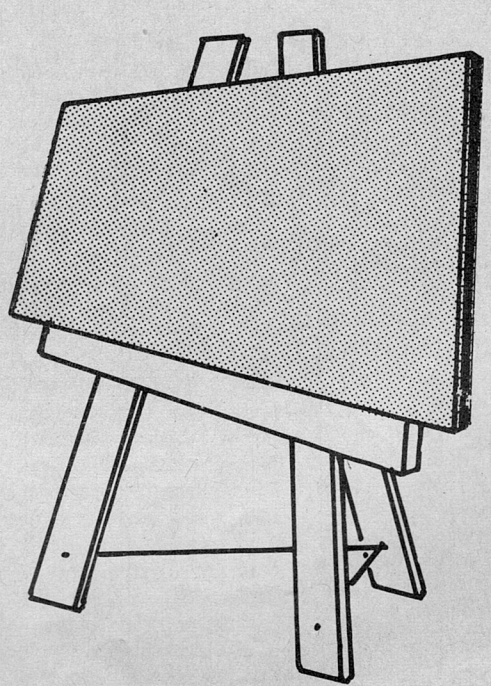


status flags (condition codes)

Status Flags = Condition Codes Key

C + carry **+ overl**

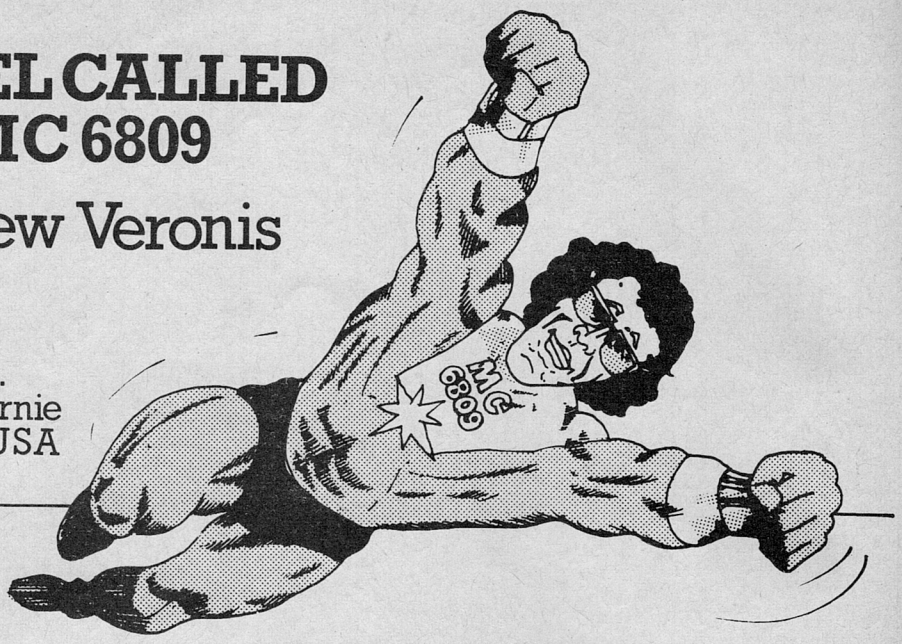
```
0090¦ A5 03                     LDA MAXBYTES+1  ;BUFFER SKIPPING
0092¦ E9 00                     SBC #0
0094¦ 85 03                     STA MAXBYTES+1
0096¦ 05 02                     ORA MAXBYTES
0098¦ 60                        RTS
0099¦
0099¦                   DONE    PUSH NBYTES
006A* 00
0063* 00
004E* 00
0030* 00
0099¦ A5 07             #       LDA NBYTES+1
009B¦ 48                #       PHA
009C¦ A5 06             #       LDA NBYTES
009E¦ 48                #       PHA
009F¦                           PUSH RETURN     ;BACK TO PASCAL
009F¦ A5 01             #       LDA RETURN+1
00A1¦ 48                #       PHA
00A2¦ A5 00             #       LDA RETURN
00A4¦ 48                #       PHA
00A5¦ 60                        RTS
00A6¦
00A6¦                           .END
PAGE -   4  GETTEXT   FILE:AFOREIGN  SYMBOLTABLE DUMP


AB - Absolute     LB - Label     UD - Undefined     MC - Macro
RF - Ref          DF - Def       PR - Proc          FC - Func
PB - Public       PV - Private   CS - Consts

BUFFER   AB 0004¦  CHARS    AB 0008¦  CR       LB 0058¦  DATA     AB C0AF¦  DOIT     LB 0079¦  DONE     LB 0099¦  GETTEXT  FC ----
KEYBOARD AB C000¦  KEYRESET AB C010¦  MAXBYTES AB 0002¦  NBYTES   AB 0006¦  POP      MC ----¦  PUSH     MC ----¦  RETURN   AB 0000
SKBYTES  AB 000A¦  SKCOUNT  AB 000C¦  SKIP1    LB 0083¦  SKIP2    LB 0089¦  SPACES   LB 0049¦  STATUS   AB C0AE¦  STOREONE LB 006E
STORIT   LB 0067¦  WAIT     LB 002B¦
PAGE -   5  GETTEXT   FILE:AFOREIGN


Current minimum space is 9125 words

Assembly complete:       156 lines
     0   Errors flagged on this Assembly
```



A MARVEL CALLED THE MC 6809

Dr Andrew Veronis

President ACFA Inc.
PO Box 1070 Glen Burnie
Maryland 21061 USA



THE MARVEL CALLED MC6809

I am sure that most of us have heard of the MC6809 by now. In this article, I will attempt to introduce you to this wonderful microprocessor.

MC6809 FEATURES

The MC6809 is an advanced processor within the M6800 family offering greater throughput, improved byte-efficiency, and increased adaptability to various software disciplines. These include position independence, re-entrancy, recursion, block structuring, and High Level Language generation. It is compatible with all M6800 peripherals, and is software compatible with the MC6800 at source code level.

This product preview stresses the improvements inherent in the MC6809. The basic operation of the MPU is similar to the MC6800. The M6800 Microcomputer System Design Data manual is a recommended source of information for those not familiar with the MC6800.

HARDWARE FEATURES

The MC6809 adds a third Interrupt Request input, three processor status outputs, and an optional internal clock system to the Control/Status signals available on the MC6800. Signals eliminated (or not required) include Valid Memory Address, Data Bus Enable, and Three-State Control.

For internal clock use, an external crystal is connected between EXTAL and XTAL. A synchronization signal is available at the ENABLE output to be used as a system clock. This signal operates at the basic processor frequency, and is normally connected to the Enable (02) inputs of M6800 family peripherals.

a READY input is provided to inhibit transitions of the clock. The duration of the clock stretch can be any integral multiple of the crystal oscillator period, up to a maximum stretch of 20 microseconds. Since a low level at the READY input inhibits any transition of the clock, it may be used for handling slow memory or for Direct Memory Access. It is not applicable to External Clock systems, and should be tied to a logic one when not used.

The External Clock mode of the MC6809 is particularly useful when it is desired to synchronize the processor to an externally generated signal. The external clock generator need provide an output only at the basic MPU frequency since the internal frequency dividers of the MC6809 are not used in the external clock mode. ENABLE is an input in this mode, as is EXTAL.

A low level on the HALT input causes the MPU to halt at the end of the present instruction, and remain halted indefinitely without loss of data, until the HALT line is driven high. When the MPU is halted, the BA output is driven high to acknowledge the halt, and the Address Bus and Data Bus drivers are made high-impedence. While halted, the MPU cannot respond to interrupt requests, nor can it be released from reset.

A Fast Interrupt Request (FIRQ) is added to the IRQ and NMI inputs available on the MC6800. When a logic zero is recognised at this input, the MC6809 places only the Program Counter and Condition Code Register on the stack prior to accessing the FIRQ vector locations to obtain the starting location of the FIRQ service routine. Either an IRQ or Non-Maskable Interrupt stacks the contents of all registers (same as MC6800) prior to vectoring. The three interrupt inputs have separate vector locations, and are prioritzed (IRQ lowest, FIRQ next, NMI highest).

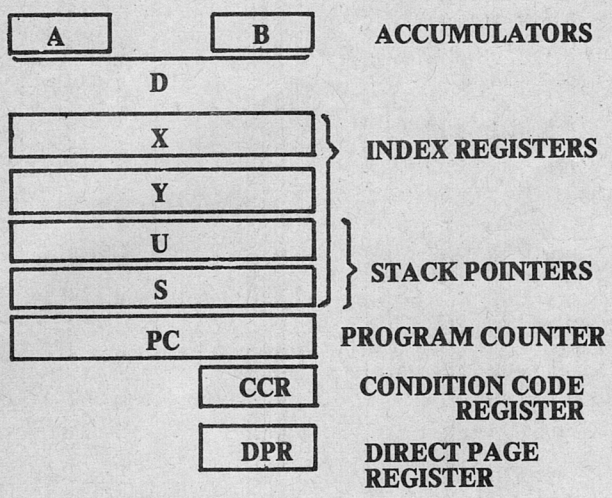
An Interrupt Acknowledge output (IACK) is provided to indicate that a vector is being fetched as a result of a RESET, a Software Interrupt (SWI, SWI1, or SW12), or recognition of NMI, IRQ, or FIRQ. This output allows the program counter to be loaded according to the interrupting device, thus providing full vectored interrupt handling.

A Last Instruction CYCLE OUTPUT (LIC) is activated during the last cycle of any instruction. This signifies that the next processor cycle will be an opcode fetch.

A Processor Busy signal (BUSY) is incorporated to facilitate multiprocessor applications. This signal is asserted during MPU Read-Modify-Write instructions, allowing the designer to insure that flags being modified by one processor are not accessed by another simultaneously. The signal is also asserted during execu-

tion of double byte instructions and when using the Indirect Addressing modes.

PROGRAMMING MODEL



PROGRAMMING MODEL

The MC6809 adds three registers to the complement available in the MC6800. These are Direct Page Register, the User Stack Pointer, and a second (Y) Index Register. In addition to increasing the number of registers, the MC6809 features much greater flexibility of register usage. For example, indexed addressing is available using a Stack Pointer (U or S) as the base register in addition to X or Y. Conversely, the Automatic Increment/Decrement option of indexed instructions allows X and Y to be used as Stack Pointers if desired. **The enhanced instruction set of the MC6809 also increases register uses. Exchanges, as well as transfers, between any two similar-width registers are allowed. Two's complement addition of the A, B, or D register [or an immediate value] to X, Y, U or S registers can be implemented via the Load Effective Address instruction. Any register - or set of registers - may be pushed to [pulled from] either stack with a single instruction.**

ACCUMULATORS

The A and B registers are general purpose accumulators used for arithmetic calculations and data manipulation. In general, the two registers are identical, although some special purpose instructions apply to the A register alone.

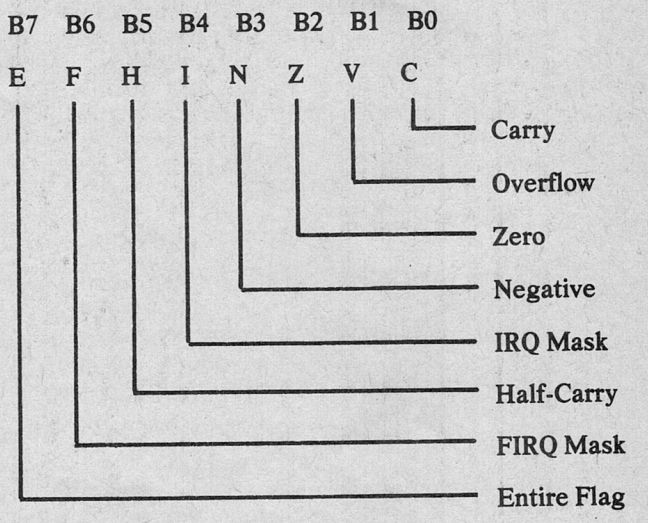
Certain instructions concatenate the A and B registers to form a single 16-bit accumulator. This is referred to as the D register, and is formed with the A Register as the Most Significant Byte.

DIRECT PAGE REGISTER

The Direct Page Register of the MC6809 serves to enhance the Direct Addressing Mode. The contents of this register appear at the higher order address outputs (A8 - A 15) during direct addressing instruction execution. This allows the Direct Page to be located at any place in memory under program control. All bits of this register are cleared during Processor RESET. This ensures 6800 compatibility.

CONDITION CODE REGISTER

The condition code register is the storage area for processor flags. Bits 0 through 5 are identical to the MC6800 CCR - Bits 6 & 7 are used for FIRQ handling.



Specifically, the E-bit controls the Return From Interrupt (RTI) instruction to insure that the proper number of registers are pulled from the Hardware Stack.

INDEX REGISTERS

The Index Registers are used in indexed mode addressing. They provide a 16-bit address to be added to an optional offset (of up to 16 bits) to form the effective address of the instruction. The X and Y registers are essentially equivalent in usage and support the same instructions. Either index register is capable of performing the same function as the MC6800 register, but can also support many additional modes of operation.

STACK POINTERS

The Hardware Stack Pointer (S) is used by the processor to automatically store machine states during sub-routines and interrupts in a manner similar to that of the MC6800. A User Stack Pointer (U) is also provided to be controlled exclusively by the program - thus allowing arguments to be passed to and from subroutines with ease.

Both U and S have the same indexed-mode addressing capabilities as the X and Y registers. This allows the MC6809 to be used effeciently as a stack processor - greatly enhancing its ability to support higher level languages. To facilitate use of the Stack Pointers in indexed-mode addressing, the registers always point to the last byte placed on the stack. (The MC6800 Stack Pointer points to the location in which the next byte will be stored).

PROGRAM COUNTER

The PC is used by the Processor to point to the next instruction to be executed by the processor. Program Counter Relative addressing is allowed on the MC6809, effectively allowing the PC register to be used as an index register. This allows Position Independent Code to be written with greater ease than with other 8-bit processors.

ADDRESSING MODES

The MC6809 retains the Addressing modes available with the MC6800, and adds long relative branches, 16 variations of indexed addressing, program counter relative and extended indirect addressing.

The enhancements to indexed addressing include auto (post) increment, auto (pre) decrement, indexing with 0, 5, 8, or 16-bit two's complement offsets, and indexing with an accumulator as an offset. Also, any of these modes may have one additional level of indirection applied. Any of the four index register (X, Y, U or S) may be used as the base Register for the indexed addressing modes.

INHERENT, ACCUMULATOR, IMMEDIATE, DIRECT, EXTENDED

These modes are effectively the same as those available with the MC6800. (The Direct Mode utilizes the DPR to form the most significant address byte as explained previously).

INDEXED

The indexed mode of addressing has several options which are selected by the byte following the opcode (postbyte). Bits 5 & 6 of this post-byte are always used to select the pointer register (X, Y, U, or S). Other options are:

Zero Offset, +4 Bits. - This option uses Bit 4 of the Postbyte as a sign bit and Bits 0 thru 3 as a constant offset. It is a minimum two-byte instruction.

Constant Offset, +7 Bits. - This option designates the byte after the Postbyte as a two's complement offset. It is a minimum four-byte instruction. (Opcode + Postbyte + 2 Byte Offset).

Accumulator Offset - This option designates the A, B, or D register as a two's complement offset. It is a minimum two-byte instruction.

In all cases, the offset is temporarily added to the contents of the selected pointer register to form an effective address.

INDEXED INDIRECT

Except for the +4 Bit Constant Offset and the Auto Increment/Decrement by one, all Indexed Addressing modes may be used with an additional level of indirection. Thus, the address formed by adding the offset to the selected pointed register designates a location containing the effective address of the operand data. Bit 4 of the Postbyte is used to select the Indexed Indirect mode . (Note that this bit is used as a sign bit in the +4 bit constant offset mode). The number of bytes for a given instruction is the same for either Indexed or Indexed Indirect addressing modes.

RELATIVE

Relative addressing involves adding a signed constant to the contents of the program counter. When used in conjunction with a Branch Instruction, this sum becomes the new Program Counter content if the branch is taken. (If the branch is not taken, the PC advances to the next instruction.)

Relative addressing differs from that contained in the MC6800 due to two important additions. The first of these is that the offset (signed constant) can be either +7 bits or +15 bits in length. This allows the program to branch to any location in the memory field.

The second important addition to the Relative mode is that it no longer is limited to Branch Instructions. An Effective Address - which retains the position-independent nature of Relative Addressing - may be formed by adding a +7 bit or +15 bit offset to the program counter. This is, in effect, an Indexed Addressing mode with one of two specific Postbytes. (The optional Postbytes allow selection of +7 or +15 bit offset.) Examples of its use would be:

```
2015 LDA -$3E,PC          2108 LDA 211F,PC

2015 A6 OPCODE            2018 A6 OPCODE
2016 8C POSTBYTE          2019 8D POSTBYTE
2017 C2 OFFSET            201A 21 OFFSET (MSB)
2018    NEXT INST         201B 1F OFFSET (LSB)
                          201C    NEXT INST.

1FDA    DATA
                          413B    DATA
```

Note that the offset is added to the new value of the program counter, i.e., the location of the next instruction.

RELATIVE INDIRECT

This addressing mode is, in effect, Indexed Indirect with the Program Counter used as an index register. One or two bytes (optional) following the Postbyte are used to provide a ±7 byte or ±15 bit offset. This signed number is added to the contents of the program counter, forming a pointer to consecutive locations in memory which contain the new effective address. Examples are:

```
2015 LDA -$3E,PC          2018 LDA 211F,PC

2015 A6 OPCODE            2018 A6 OPCODE
2016 9C POSTBYTE          2019 9D POSTBYTE
2017 C2 OFFSET            201A 21 OFFSET(MSB)
2018                      201B 1F OFFSET
                          201C

1FDA 01 NEW
1FDB 00 EA                413B 03 NEW
                          413C 00 EA
0100    DATA
                          0300    DATA
```

EXTENDED INDIRECT

This addressing mode is actually another option of Indexed Indirect Addressing. In this case, the two bytes following the Postbyte are used as a pointer to consecutive locations in memory which contain the new effective address. An example is:

```
201C LDA C200

201C A6 OPCODE
201D 9F POSTBYTE
201E C2
201F 00 POINTER
2020

C200 00 NEW
C201 80 EA

0080    DATA
```

ABSOLUTE INDIRECT

This mode is exclusively used for Restart and Interrupt vectoring. Servicinh of these conditions

involves fetching the contents of an exact location in memory to be loaded into the Program Counter.

INSTRUCTION SET

A complete listing of the Executable instructions is contained in Tables 1 through 5. Some of the more unique instructions include Load Effective Address, Synchronization with Interrupt, and Exchange Registers. These, along with other not available on the MC6800, are detailed in the following paragraphs.

Load Effective Address - Besides it obvious use, this instruction represents a convenient means of modifying any of the pointer (X,Y,S, or U) registers. The processor forms an Effective Address as dictated by the addressing mode, then loads this value into the designated register rather than outputting the data on the address lines. As an example, LEAX A,X will add the signed number contained in the A register to the contents of the X register, then place this result into the X register.

Synchronize with Interrupt - This instruction causes processing to discontinue until an Interrupt Input (NMI, IRQ, or FIRQ) is activated. When an interrupt occurs, and the associated masks clear, the processor services the interrupt normally. If the interrupt mask is set when the interrupt occurs, the processor exits the Synch Mode by continuing to the next instruction. The Bus Available output is activated during the Synch Mode. The instruction is particularly useful for synchronizing the program with a peripheral which uses or supplies data.

Exchange and Transfer Registers - Both these instructions utilize an immediate byte to define the source and destination registers. The only restriction is that both source and destination must be similarly sized registers.

Push/Pull Register(s) - These instructions also use an immediate byte to designate whether the register assigned to a particular bit is to be affected. Thus, a Push instruction followed by a byte containing a 'one' in bit 7 causes the Program Counter to be pushed onto a stack. 1 - 8 registers can be pushed or pulled with a single instruction.

Sign Extend - This instruction causes all bits in the A register to take on the value of the Most Significant Bit of the B register.

CWAI - This instruction is similar to the Wait for Interrupt used with the MC6800, but includes an immediate byte to clear condition codes if desired. The CWAI instruction can be used with any of the three interrupt lines, even though CWAI stacks all registers (except S). After stacking is complete, Bus Available is activated and the processor idles until an interrupt occurs.

Those familiar with the MC6800 will note that some instructions are missing from the complement available with the MC6800. Provisions have been made to perfrom such operations in alternative ways when required. An example might be Decrement X. This will not often be neede with the MC6800 kldue to the Auto-Decrement option with Indexed Addressing. If needed, however, the operation can be accomplished with an LEAX -1,X instruction. Likewise, the MC6800 instructions to Clear/Set various condition codes are replaced with ANDCC/ORCC. In this manner, the MC6809 uses fewer instruction mnemonics (59 v's 72) than the MC6800, yet is fully software compatible, as well as being considerably more powerful.

Table 1.

— 8-BIT ACCUMULATOR AND MEMORY INSTRUCTIONS

Mnemonic(s)	Operation
ADCA, ADCB	Add memory to accumulator with carry
ADDA, ADDB	Add memory to accumulator
ANDA, ANDB	And memory with accumulator
ASL, ASLA, ASLB	Arithmetic shift of accumulator or memory left
ASR, ASRA, ASRB	Arithmetic shift of accumulator or memory right
BITA, BITB	Bit test memory with accumulator
CLR, CLRA, CLRB	Clear accumulator or memory location
CMPA, CMPB	Compare memory from accumulator
COM, COMA, COMB	Complement accumulator or memory location
DAA	Decimal adjust A-accumulator
DEC, DECA, DECB	Decrement accumulator or memory location
EORA, EORB	Exclusive or memory with accumulator
EXG R1, R2	Exchange R1 with R2 (R1, R2 = A, B, CC, DP)
INC, INCA, INCB	Increment accumulator or memory location
LDA, LDB	Load accumulator from memory
LSL, LSLA, LSLB	Logical shift left accumulator or memory location
LSR, LSRA, LSRB	Logical shift right accumulator or memory location
MUL	Unsigned multiply (A x B → D)
NEG, NEGA, NEGB	Negate accumulator or memory
ORA, ORB	Or memory with accumulator
ROL, ROLA, ROLB	Rotate accumulator or memory left
ROR, RORA, RORB	Rotate accumulator or memory right
SBCA, SBCB	Subtract memory from accumulator with borrow
STA, STB	Store accumulator to memory
SUBA, SUBB	Subtract memory from accumulator
TST, TSTA, TSTB	Test accumulator or memory location
TFR, R1, R2	Transfer R1 to R2 (R1, R2 = A, B, CC, DP)

NOTE: A, B, CC, or DP may be pushed to (pulled from) either stack with PSHS, PSHU, (PULS, PULU) instructions.

Table 2.

— 16-BIT ACCUMULATOR AND MEMORY INSTRUCTIONS

Mnemonic(s)	Operation
ADDD	Add memory to D accumulator
CMPD	Compare memory from D accumulator
EXG D, R	Exchange D with X, Y, S, U or PC
LDD	Load D accumulator from memory
SEX	Sign Extend B accumulator into A accumulator
STD	Store D accumulator to memory
SUBD	Subtract memory from D accumulator
TFR D, R	Transfer D to X, Y, S, U or PC
TFR R, D	Transfer X, Y, S, U or PC to D

Table 3.

— INDEX REGISTER/STACK POINTER INSTRUCTIONS

Mnemonic(s)	Operation
CMPS, CMPU	Compare memory from stack pointer
CMPX, CMPY	Compare memory from index register
EXG R1, R2	Exchange D, X, Y, S, U, or PC with D, X, Y, S, U or PC
LEAS, LEAU	Load effective address into stack pointer
LEAX, LEAY	Load effective address into index register
LDS, LDU	Load stack pointer from memory
LDX, LDY	Load index register from memory
PSHS	Push any register(s) onto hardware stack (except S)
PSHU	Push any register(s) onto user stack (except U)
PULS	Pull any register(s) from hardware stack (except S)
PULU	Pull any register(s) from hardware stack (except U)
STS, STU	Store stack pointer to memory
STX, STY	Store index register to memory
TFR R1, R2	Transfer D, X, Y, S, U or PC to D, X, Y, S, U or PC
ABX	Add B accumulator to X (unsigned)

Table 4.

— BRANCH INSTRUCTIONS

Mnemonic(s)	Operation
BCC, LBCC	Branch if carry clear
BCS, LBCS	Branch if carry set
BEQ, LBEQ	Branch if equal
BGE, LBGE	Branch if greater than or equal (signed)
BGT, LBGT	Branch if greater (signed)
BHI, LBHI	Branch if higher (unsigned)
BHS, LBHS	Branch if higher or same (unsigned)
BLE, LBLE	Branch if less than or equal (signed)
BLO, LBLO	Branch if lower (unsigned)
BLS, LBLS	Branch if lower or same (unsigned)
BLT, LBLT	Branch if less than (signed)
BMI, LBMI	Branch if minus
BNE, LBNE	Branch if not equal
BPL, LBPL	Branch if plus
BRA, LBRA	Branch always
BRN, LBRN	Branch never
BSR, LBSR	Branch to subroutine
BVC, LBVC	Branch if overflow clear
BVS, LBVS	Branch if overflow set

Table 5.

— MISCELLANEOUS INSTRUCTIONS

Mnemonic(s)	Operation
ANDCC	AND condition code register
CWAI	AND condition code register, then wait for interrupt
NOP	No operation
ORCC	OR condition code register
JMP	Jump
JSR	Jump to subroutine
RTI	Return from interrupt
RTS	Return from subroutine
SWI, SWI2, SWI3	Software interrupt (absolute indirect)
SYNC	Synchronize with interrupt line

Reach the people who matter with the LIVERPOOL SOFTWARE GAZETTE

The microcomputer magazine that is

An informal poll of our readership (3 samples) showed that they were well educated, normal, responsible people who buy things.

It therefore obviously pays to advertise . . .
Ridiculously reasonable rates high quality editorial and production standards make us unbeatable media in which to advertise **your** product, however mythical it may be. Advertise that Z-8000 board here, the £325 1200 CPS impact Printer, or Hard Disk sub-system for SC/MP.

Notes		
Full page	(17.5 cm x 24 cm)	£180.00
Half Page	(Upright 8.5 cm x 24 cm)	£95.00
	(Landscape 17.5 cm x 11.75 cm)	£95.00
Quarter Page	(8.5 cm x 11.75 cm)	£52.00

Agency discount 10%

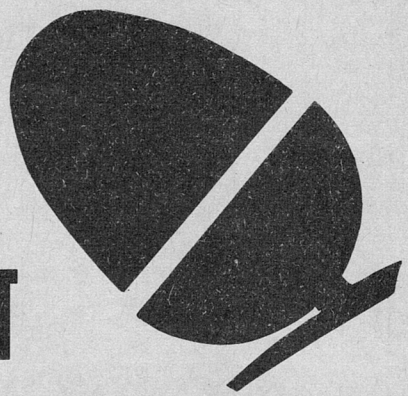
Copy date for the July issue is June 20
Copy date for the September issue is August 8

ALL ADVERTISERS RECEIVE 50 COPIES OF THE ISSUE THEY ADVERTISE IN, FREE OF CHARGE

14 Castle Street
Liverpool L2 OTA
Tel: 051-227 2535/6/7/8

A number processer on the ACORN

LAURENCE HARDWICK



The MM57109 from National is a processor intended for use in number orientated applications. The processor can be used as a peripherals device attached to a microprocessor system, or as the heart of a stand alone system. This series of two articles describes the interfacing of the MM57109 to an Acorn System 1.

This article describes the hardware interface and general considerations of the software involved. The second article describes the complete software required to turn the Acorn into quite a sophisticated programmable calculator, while specific reference is made to the Acorn system, the hardware and software should be of interest to anyone considering using the MM57109 as a microporcessor peripheral.

The MM57109

The MM57109 obeys instructions in much the same way as a calculator obeys the instructions given by the keys you press. It has full scientific calculator instructions in reverse polish notation with numbers in floating point or scientific form. The numbers have an 8 digit mantissa and 2 digit exponent and are stored in a 4 register stack or in the one memory register. Functions available include trigonometric, logarithmic and arithmetic Y^X, e^X, $\times$, etc the full list of possible instructions is given in table 1.

Hardware

The MM57109 presents a few problems to the designer because of its power supply and signal buffering requirements. The processor requires a power supply of 9v at about 15 MA and this could be a problem in the usual 0-5v microcomputer system. The circuit shown in figure 1 requires only a single 5v rail and includes an inverter which provides -3v which is sufficient to run the 57109 processor.

The interface signals require various different buffering arrangements, and these are shown in the circuit. The interface to the microprocessor uses two 8-bit parallel ports, this is via an INS8154 on the Acorn system, as all handshaking is done by software any two 8 bit parallel ports or latches may be used.

Interface Signals

The output port from the host microprocessor provides the following signlas to the 57109.

BIT 7	Power On Reset	Low to reset
BIT 6	Hold	Low to hold
BIT 5-0	Instruction codes	+ve logic

The input port to the host system has the following signals on it:

BIT 7	RDY	High when ready for next instruction
BIT 6	R/W	High when 57109 outputs data
BIT 5	BR	High when jump occurs
BIT 4	ERROR	High when error occurs
BIT 3-0	DATA	Output from 57109 +ve logic

OPERATION

After sending the 57109 a reset pulse and waiting for the R/W line to go low tye number processor is ready to receive instruction codes. The 57109 requests each instruction by sending th RDY line high, if the hold line is low or goes low while RDY is high, RDY will go low again. The instruction code must be valid on lines L_1-I_6 of the 57109 at the time of the high to low transition of RDY.

The instruction codes are listed in table 1. In the program listed the instructions are held in the form of a table from location 0300 upwards in memory.

In the process of executing its instructions the 57109 may send other signals to the host computer.

The R/W line will go high in response to the OUT instructions and signifies that data is available on pins D1 - D4 of the 57109.

Each time the R/W signal goes high this data should be read and stored in the hosts memory to be displayed when required. The OUT instruction is a two word instruction and the second word can be used by the host system to specify area of memory in which the data should be stored.

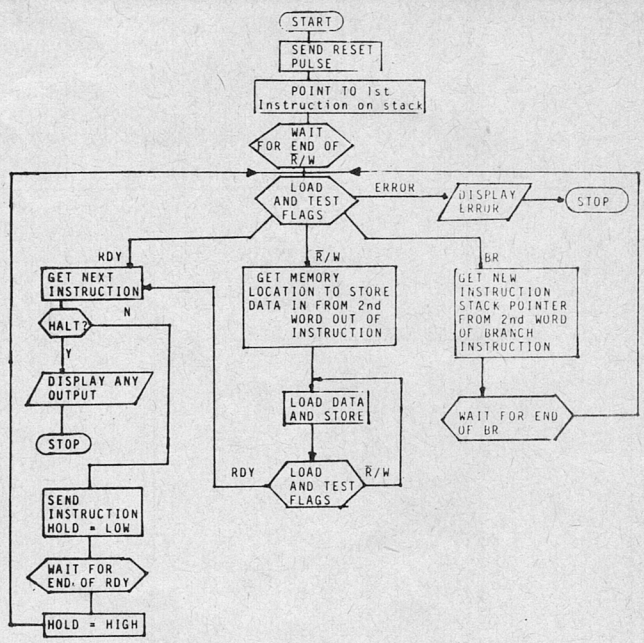
The BR line signifies that a JUMP should be made as a result of a conditional or unconditional branch instruction. All branch instructions are two words long the second word should be loaded by the host system as the address in the instruction table to jump by the host system as the address in the instruction table to jump to.

The error line goes high when an error has occured, in the program listing this causes the word 'ERROR' to be displayed. Other software could be written to trap these errors and attempt to interpret them. The possible errors will occur when calculating.

LNX or LOGX when X__0

Any result 10^{-99} or 10^{100}
Tan 90°, 270°, 450°, etc
SIN, COS, TAN X when X 9000°
Sin^{-1} , Cos^{-1} , X when 1 x 1 1 or 1 x 1 10-50
SORT X when X 0
Division by zero
When outputting a number in floating point mode is too large for this mode.

Taking each of these signals into account the flow chart of the simple interface program is shown here:



The program given in listing 1 takes the instructions from a table at 0300 upwards in memory and sends these to the MM57109. When the 57109 outputs data it is stored, and displayed when the halt code 0F is reached in the instruction table, the second byte of the OUT instruction must always be 30 when using this program. The program does not allow numbers to be ouput in scientific notation and attempting to OUTPUT any number larger than 99999999 will cause an error.

The programs for the 57109 given here demonstrates the ways in which numbers can be input, and some of the capabilities of the processor.

PROGRAM 1 PRINT P1

LOCATION	INSTRUCTION	CODE (HEX)	COMMENTS
0300	P1	0D	ENTER P1
0301	OUT (30)	16 30	OUTPUT
0303	HALT	0F	END AND DISPLAY

PROGRAM 2 FACTORIAL 8

0300	8	08	ENTER NUMBER
0301	X - M	1B	
0302	1	01	
0303	X - M	1B	
0304	X x M - M	20 3B	
0306	X - M	1B	DECREMENT M and
0307	DBNZ	1A 03	BRANCH TO 03 IF M ≠0
0309	OUT (30)	16 30	OUTPUT N!
030B	HALT	0F	END AND DISPLAY

PROGRAM 3 PRINT -100

0300	1	01	
0301	+/-	0C	Change SIGN
0302	EE	0B	ENTER EXPONENT
0303	2	02	
0304	OUT (30)	1630	OUTPUT
0306	HALT	0F	END AND DISPLAY

TABLE 1

INSTRUCTION CODES

2nd DIGIT	1st DIGIT 0	1	2	3
0	0		*INV	$X_x - Y$
1	1	*JMP(X=0)	ENTER	e^x
2	2	*JMP(X<0)	FP-SN	10^x
3	3	*JMP(|X|<1)	ROL	X^2
4	4	*JMP(E=1)	SIN(SIN^{-1})	X
5	5	*JMP	COS(COS^{-1})	LNX
6	6	*OUT	TAN(TAN^{-1})	$LOG_{10}X$
7	7			$1/X$
8	8			
9	9	*IBNZ		+(M+X-M)
A	DP	*DBNZ		-(M-X-M)
B	EE	X-M	E+0	X(MxX-M)
C	+/-	X-M	RAD-DEG	1(M/X-M)
D		M-X	DEG-RAD	X^y
E		$X_m \leftarrow$	POP	
F	HALT	$X_m \rightarrow$	CLEAR	NOP

NOTES

* TWO WORD INSTRUCTIONS

() FUNCTION IN BRACKETS GIVEN BY INV (FN)

$\underline{()}$ CONDITION TRUE FOR JUMP

E ERROR FLAG

POP STACK OPERATION Y→X; Z→Y; T→Z; Ø→T

ROLL STACK OPERATION Y→X→T→Z→Y

$X_{m\rightarrow}^{\leftarrow}$ SHIFT DIGITS LEFT/RIGHT ONE PLACE

IBNZ M=M+1 JMP $\underline{(M\neq 0)}$

DONZ M=M-1 JMP $\underline{(M\neq 0)}$

FP-SN SWITCH MODE BETWEEN SCIENTIFIC AND FLOATING POINT

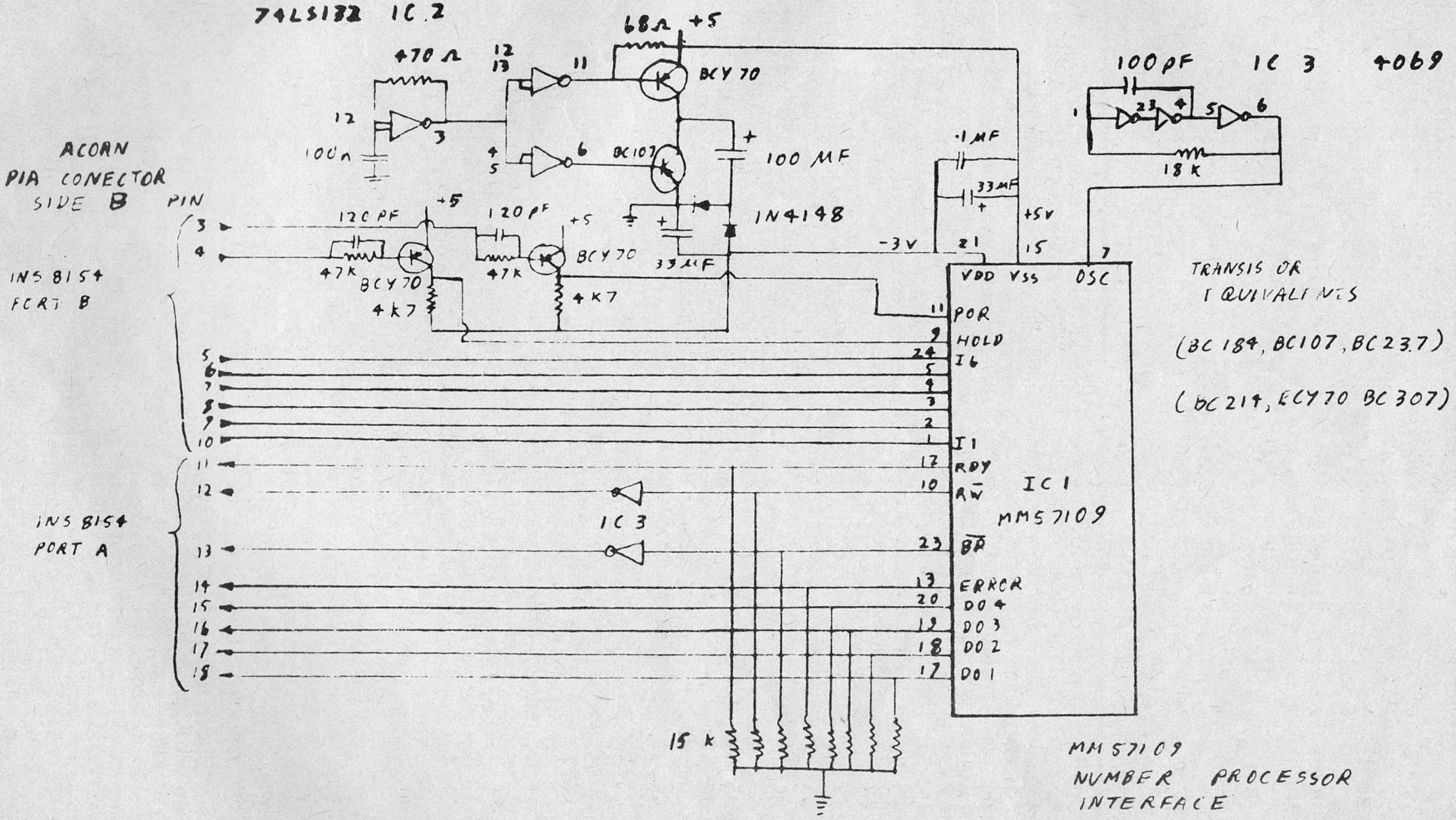
ALL SINGLE NUMBER (UNARY) OPERATORS
ACT ON THE X ANSWER IN X

ALL TWO NUMBER (BINARY) OPERATORS
ACT ON X? Y ANSWER IN X.

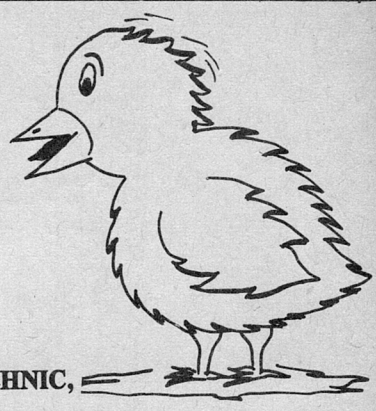
ACT ON X, Y ANSWER IN X

```
          ACORN 6502 Assembler   Page 01
          LISTING 1

0010:                    **********************
0020:                    * NUMBER PROCESSOR    *
0030:                    *      INTERFACE     *
0040:                    *                    *
0050:                    *   MM57109          *
0060:                    **********************
0070: 0200           NCI    ORG    $0200
0080: 0200           DISREG *      $0010      Acorns display register
0090: 0200           SIGN   *      $0030        Place to put number
0100: 0200           NUMBER *      SIGN   +01          from 57109
0110: 0200           DPOINT *      NUMBER
0120: 0200           RESET  *      $FF04     Acorn monitor calls
0130: 0200           TABLE  *      $FFEA
0140: 0200           DISPLY *      $FE0C
0150: 0200           STACK  *      $0300      57109 instruction stack
0160: 0200           PIA    *      $0900      INS8154 at HEX 0900
0170: 0200           DRA    *      PIA    +20
0180: 0200           DRB    *      PIA    +21
0190: 0200           DDRA   *      PIA    +22
0200: 0200           DDRB   *      PIA    +23
0210: 0200           TESTA  *      PIA
0220: 0200           SETB   *      PIA    +18
0221:                  ***
0222:                  *  START SEND POR
0223:                  ***
0230: 0200           CLEARB *      PIA    +08
0240: 0200 A9 FF            LDAIM $FF
0250: 0202 8D 23 09         STA   DDRB
0260: 0205 A9 40            LDAIM $40
0270: 0207 8D 21 09         STA   DRB
0280: 020A A2 0A            LDXIM $0A
0290: 020C CA        WAIT   DEX
0300: 020D D0 FD            BNE   WAIT
0310: 020F 8D 1F 09         STA   SETB   +07
0320: 0212 AD 06 09  EWRITE LDA   TESTA  +06
0330: 0215 30 FB            BMI   EWRITE   waiting for end of write
0350: 0217 AD 20 09  TEST   LDA   DRA     load and test for flags
0360: 021A 30 0B            BMI   RDY     ready for next code
0370: 021C 2A               ROLA
0380: 021D 30 22            BMI   WRITE    output a number
0390: 021F 2A               ROLA
0400: 0220 30 37            BMI   JUMP    do a jump
0410: 0222 2A               ROLA
0420: 0223 30 67            BMI   ERROR   display error message
0430: 0225 10 F0            BPL   TEST
0440: 0227 BD 00 03  RDY    LDAAX STACK    get next code off stack
0450: 022A E8               INX
0460: 022B C9 0F            CMPIM $0F
0470: 022D D0 03            BNE   NOTEND
0480: 022F 4C 64 02         JMP   OUTPUT   display number and end
0490: 0232 09 C0     NOTEND ORAIM $C0
0500: 0234 8D 21 09         STA   DRB
0510: 0237 AD 07 09  ERDY   LDA   TESTA  +07
0520: 023A 30 FB            BMI   ERDY      wait for end of ready
0530: 023C 8D 0E 09         STA   CLEARB +06    and assert hold
0540: 023F 10 D6            BPL   TEST    NCI   ACORN 6502 Assembler   Page 0
0550: 0241 BC FF 02  WRITE  LDYAX STACK  -01  get pointer
```

```
0560: 0244 6A        SAVE   RORA            to store number at
0570: 0245 99 00 00         STAAY $00
0580: 0248 AD 06 09  ERW    LDA    TESTA  +06
0590: 024B 30 FB             BMI    ERW
0600: 024D AD 20 09  AGAIN  LDA    DRA
0610: 0250 30 D5             BMI    RDY      ready pulse means end of write
0620: 0252 2A                ROLA
0630: 0253 10 F8             BPL    AGAIN
0640: 0255 C8                INY
0650: 0256 18                CLC
0660: 0257 90 EB             BCC    SAVE
0670: 0259 BD FF 02  JUMP   LDAAX  STACK  -01   get place to jump to
0680: 025C AA                TAX
0690: 025D AD 05 09  EJUMP  LDA    TESTA  +05
0700: 0260 30 FB             BMI    EJUMP
0710: 0262 10 B3             BPL    TEST
0720: 0264 A5 31     OUTPUT LDA    DPOINT  convert and display number
0730: 0266 E9 44             SBCIM  $44     adjust DP position
0740: 0268 85 31             STA    DPOINT
0750: 026A A5 30             LDA    SIGN
0760: 026C 0A                ASLA
0770: 026D 0A                ASLA
0780: 026E 0A                ASLA
0790: 026F 85 10             STA    DISREG
0800: 0271 A2 07             LDXIM  $07
0810: 0273 B5 31     MORE   LDAAX  NUMBER
0820: 0275 29 0F             ANDIM  $0F
0830: 0277 A8                TAY
0840: 0278 B9 EA FF          LDAAY  TABLE   get font from acorn monitor
0850: 027B C6 31             DEC    DPOINT
0860: 027D D0 02             BNE    NOPOIN
0870: 027F 09 80             ORAIM  $80      put in a DP
0880: 0281 95 10     NOPOIN STAAX  DISREG
0890: 0283 CA                DEX
0900: 0284 D0 ED             BNE    MORE
0910: 0286 20 0C FE          JSR    DISPLY
0920: 0289 4C 04 FF          JMP    RESET
0930: 028C A0 07     ERROR  LDYIM  $07
0940: 028E B9 9D 02  ELOOP  LDAAY  EMESS
0950: 0291 99 10 00          STAAY  DISREG
0960: 0294 88                DEY
0970: 0295 10 F7             BPL    ELOOP
0980: 0297 20 0C FE          JSR    DISPLY
0990: 029A 4C 04 FF          JMP    RESET
1000: 029D 00        EMESS  =      $00
1010: 029E 79                =      $79      E
1020: 029F 50                =      $50      r
1030: 02A0 50                =      $50      r
1040: 02A1 5C                =      $5C      o
1050: 02A2 50                =      $50      r
1060: 02A3 00                =      $00
1070: 02A4 00                =      $00
ID
```


74LS132 IC 2
68Ω +5
470Ω
12
13
11
BCY 70
12
3
100n
4
5
6
BC107
100 MF
+
ACORN
PIA CONECTOR
SIDE B PIN
120 PF
+5
120 PF
+5
IN4148
3
4
47k
BCY 70
47k
BCY 70
33µF
4k7
4k7
INS 8154
PORT B
INS 8154
PORT A
IC 3
15 k
-3V
21
+5V
15
7
VDD
VSS
OSC
POR
HOLD
I6
I1
RDY
RW
IC 1
MM57109
BR
ERROR
DO 4
DO 3
DO 2
DO 1
100 pF
IC 3
4069
1
23
4
5
6
18 K
·1 MF
33 MF
+
TRANSIS OR
EQUIVALENTS
(BC 184, BC107, BC237)
(BC 214, BCY70 BC 307)
MM 57109
NUMBER PROCESSOR
INTERFACE
L HARDWICK

Architectural software on the cheap



PAUL COATES,
LECTURER AT DEPARTMENT OF ARCHITECTURE, LIVERPOOL POLYTECHNIC, 53, VICTORIA STREET, LIVERPOOL L1 6EY.

LIVERPOOL Polytechnic Department of Architecture ordered its first micro (a 7k PET) in February 1978. This arrived in July of that year. Since then the work of the Department has developed and we now have two 32k PETs, Commodore disk and printer and a Hewlett Packard Plotter. We are now approaching the point where we can offer a reasonable service to our students and staff, with two part-time research assistants and 2½ years of work, and we are planning to develop a range of educational software that will be useful to all of the 20 or so schools of architecture with PETs or other similar machines.

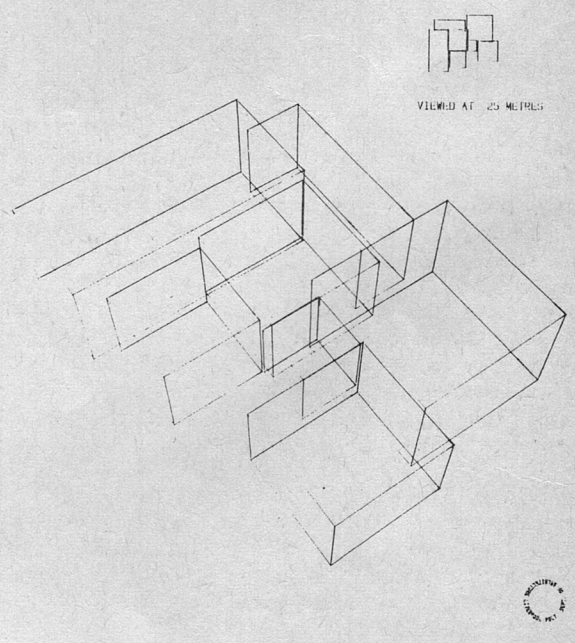
I had decided to write this article by putting together all the notes and explanations that I have written over the last two years about our aims and intentions, usually for the benefit of some worthy grant awarding body or committee. However, it would be a pity to add to the enormous literature of what people would like other people to think they're doing, and instead I will simply talk about those programs that we have working **now.**

These programs fall into two main categories. Those that are supposed to get the student to think ('teaching programs') and those that do his thinking for him (programs). The former, which I find most interesting, are harder to well to students, who not surprisingly resent being made to think by a machine. I try to point out that they don't mind being forced to think by a piece of cardboard when they are playing Monopoly, but of course it takes luck and inspiration to be instructional and amusing at the same time.

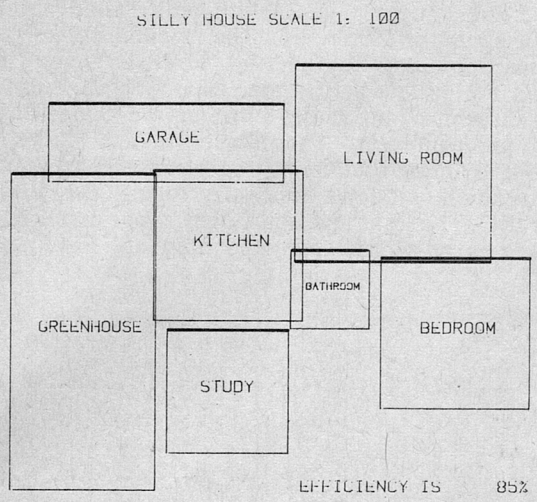
Students are reluctant to invest time and energy in using the computer unless they can get something very tangible out of it, and so I will draw a veil over the PET version of the Open University Problem Identification Game, or our general purpose Multiguess (based on the animal game) pausing only to mention our version of PEGASUS (written for PDP 11 by Mildred Shaw of Middlesex Polytechnic).

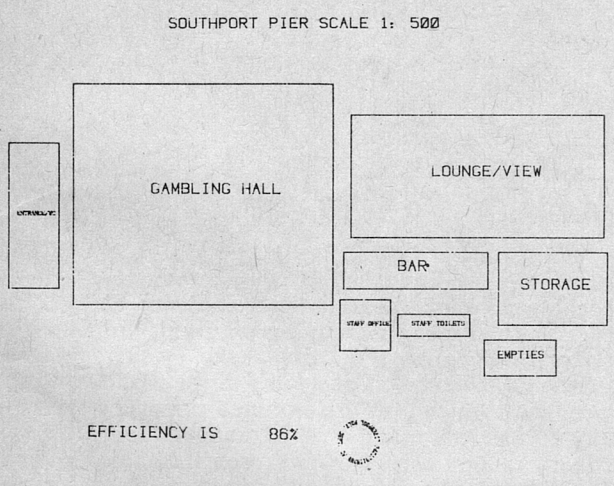
Pegasus was originally written as a psychological testing procedure but it can be used with students to explore the way they think about a problem. At the end of a half hour session they come away with a neat pair of cluster diagrams showing the results of a statistical analysis of their responses. The only other half decent teaching programs we have are Fawlty Towers and Planner. Fawlty Towers is actually quite a good game, but the intention was to use it as a way of explaining aspects of graph theory, which would be used in Planner. The hunt the dead body business (for it is on the surface a sort of Cluedo) is based on a random graph - a set of rooms which may or may not interconnect. Once the body is found there is a quiz about the plan which usually means that, unless you made some kind of map you get poor results.

Once you've got the idea across that a **plan** can be represented as a network of points and lines, which in turn can be represented in a **matrix**, then it's on to Planner. Here you define the ideal relationships between rooms on a matrix, then assemble a plan on the screen and test the efficiency (i.e. how far it corresponds to the 'ideal' relationships). This is all very easy to do (it had better be or else most of the students would walk away) using the joystick (kindly provided with software by John Stout Q.V.) and four keys. There is a menu of commands that allow you to Swap, Alter (shape), Change (position), change the Matrix, draw a perspective (Fig.1)



(of which more later) or file data on disk for use after tea, or plot out the plan with labels of room names and the efficiency (Fig.2).

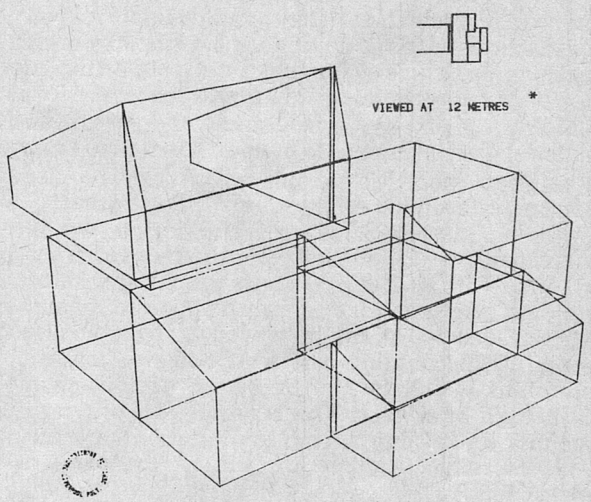




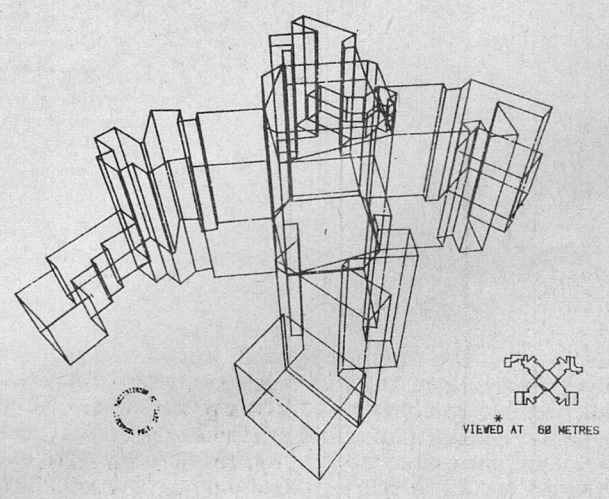
The object of the exercise is to teach students about the use (and abuse), usefulness and limitations of the connectivity matrix. This is the table that defines the 'ideal' relationships and the program allows you to set it up, try out and plan, see the result, move things around, try it again, and change an item in the matrix, try it again and so on.

Most students find it fairly amusing and quickly grasp the circularity of the argument. This years First Year did a traffic survey of the Third Floor in our building and from that we input the movement data to the matrix (this was all done with one student at the PET and the rest looking at the display on a 36" telly - interface from STACK). We then 'drew' the plan of the Third Floor on the screen and tested the efficiency. It came out as a dismal 36%, so they all tried moving things about - we got it up to 60 odd by putting the bogs in the middle of the staff room and moving the stairs next to them. In one afternoon I was able to demonstrate graphically most of the fundamentals of space planning vis a vis circulation, we had lots of chat about the problem and everyone had had a jolly time.

Apart from these game based programs we have a range of programs that cover the wide range of specialist subjects that architecture is connected with, Building Economics, Structures, Light and Heat, and graphical presentation. These are used usually by the specialist staff involved, except for the perspective program. This has been recently written by Anne Scott (our RIBA-funded research assistant) and provides a very easy way of digitising information from a plan of the building, using the plotter, and drawing perspective views (without hidden line removal). The program makes an assumption that a building can be defined as a series of vertical planes with defined top and bottom edges (which can be sloping) plus a 'roof' of freely defined points if necessary. This allows quite complex buildings to be digitised quite quickly using a user defined set of 10 keys to indicate 'wall types' (i.e. predefined top and bottom heights). A series of pitched elements disposed down a slope (as in the drawing - Fig.3) wa digitised in a few minutes.

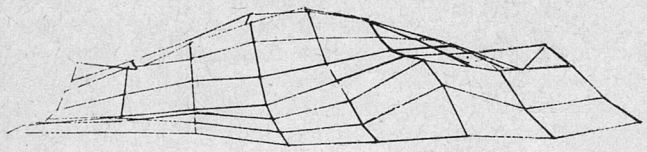


This program is without hidden line removal, but for a student with a fairly complex plan it does offer a chance to generate 20 or so views in the course of say an afternoon. (Fig.4)



The perspective routines are written as a self contained lump and can be used to provide 3-D views of a variety of objects, hence their appearance in 'Planner'.

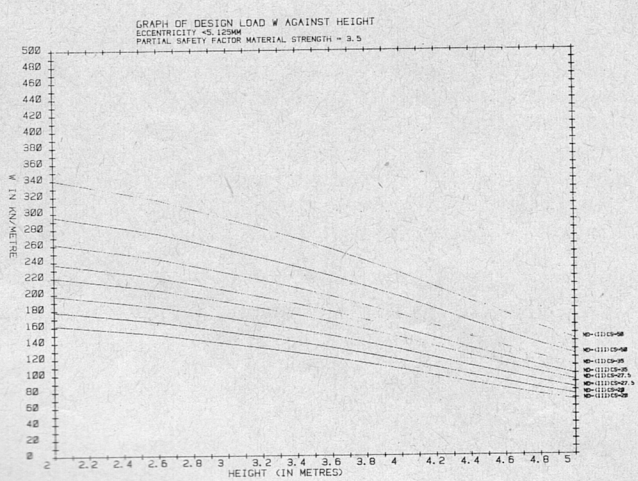
We have recently (Friday 18th) finished a 'site perspective' program using the same routines to generate an approximation to a complex curved surface (see fig.5)



using data digitised from a map on the plotter, (the view is of a part of the environs of Craigeiver Castle in the Cairngorms - with the vertical scale exagerated three times).

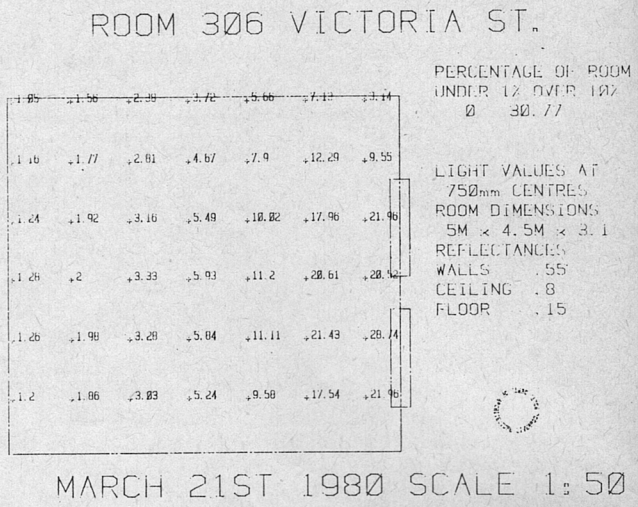
In the last six months Susan Davies (our Nuffield sponsored research assistant) and the staff responsible for teaching building economics have been working on teaching programs to demonstrate some of the subtleties of the inter-relationships between cost factors in building - geometry, glazed areas and cost of walling material for instance. We now have two programs 'Costglaze' and 'Costgeom' which can be used to demonstrate simple relationships in these areas.

In the general area of structural design, we have at the moment only one rather primitive teaching programme - Beam Tutor - which graphically shows the bending stresses in a simply supported beam, and where the supports can be moved to show the effect on the stress distribution. There are now two non teaching programs as well, one for steel beam design and one for calculated brickwork. The beam design program uses the disk for holding the tables of section geometry and will determine the best size either restrained or unrestrained.


GRAPH OF DESIGN LOAD W AGAINST HEIGHT
ECCENTRICITY =5.125MM
PARTIAL SAFETY FACTOR MATERIAL STRENGTH = 3.5
W KN/M
HEIGHT (IN METRES)

The graph (enclosed, Fig.6) was produced by the calculated brick-work program and it indicates all possible combinations of height and load for eight combinations of mortar and brick. The program is really a one-off since once all 96 available varieties have been plotted (on 12 sheets of paper for clarity) you never need to run it again (assuming you don't loose the output). These diagrams are shortly to be available to architects via the Brick Development Association.

These structural programs were written quite recently and subsequently we haven't really tried building their use into the curriculum. The few items of software in environmental engineering however have been around for quite a long time, especially the daylight program which I originally wrote on the Department's Nova minicomputer. The latest version allows the student to define the dimensions of a rectangular room and position windows on walls (using the joystick naturally). The light levels are then either displayed on the screen or plotted out (see drawing, Fig.7).



This program is used as part of the environmental science teaching in First Year, where students check light levels in their design using the program, the BRS protractors (boring), or the artificial sky Ywhich has to be seen to be believed). The quality of the results and the speed with which they are obtained convinces even the most recalcitrant that it's worth doing.

In this same area Neil Sturrock (head of Environmental Science teaching in the Department) has written two programs to calculate heat losses and loads in a structure, and the interstitial condensation risk in multi layer construction - these two are used by students simply to check results in lab wor.

The project to develop educational software is only just beginning and we hope to develop a number of properly instructional programs over the next 18 months, as opposed to the rather ad hoc collection of stuff just described.

On of the problems that I am just beginning to fact is that on the one hand you have people who can write programs and on the other teachers with specialist knowledge. It is very difficult to presuade the latter to set aside the time and generate the energy to sit down with the former, so as to write a lesson. I am convinced that in the areas like structures and accoustics, it would be of immense utility not only to the student, but also the members of staff to arrive at some clear and well documented tour through certain aspects of the subject - but can I convince the other people on the staff? Time will tell.

To end on an even less cheerful note - one of the problems we are beginning to face is tha bottle neck of only two machines with two research assistangs and 30 students all competing for time. We ordered a further

two PETs with the promise of sharing a third which would have solved the problem at least for the next few months, only to find that some mystical figure called 'audit' has stopped all micro purchases in the City. So we wait, and meanwhile write more pleading memos to the relevant committees ...

Hardware

The plotter referred to is a Hewlett Packard A4 size 7225A which interfaces directly to the PETs IEEE port. Should you need more information on how to drive plotters from PETs in Basic you could look at the article I've written for IPUG (if they ever publish it).

Over-loaded Bus Driver

COMMERCIAL MICROSOFTWARE FUNDEMENTALS

Compiled by G.P.JONES Software Dept.MICRODIGITAL LTD

Error messages and how to handle them.

SO, your machine is up and running, hardware is fine, what goes wrong now? Software! The error messages that we produced by the APPLE are in two categories, Disc based and Non Disc based.

In this edition, I will deal with the Non Disc based errors. To intercept the errors for both disc and non disc APPLESOFT programmes (Integer basic does not have this facility) make line Ø into Ø ONERR GO TO 6ØØØØ.

This tells the computer that if an error should occur, jump to line 6ØØØØ.

The first line i.e. 6ØØØØ should be: 6ØØØØER = PEEK (222)

This sets ER to the value of the error that has been encountered.

Table of Non Disc Error Codes

Code	Error
0	NEXT without FOR
16	Syntax
22	RETURN without GOSUB
42	Out of DATA
53	Illegal Quantity
69	Overflow
77	Out of memory
90	Undefined Statement
107	Bad subscript
120	Redimensional Array
133	Division by zero
163	Type mismatch
176	String too long
191	Formula too complex
224	Undefined function
254	Bad response to input
255	Control C attempted

To find out where the error has occured, use this routine.

6ØØ1Ø LINEO = PEEK (218) + (255 x PEEK (219))

How Errors Can Occur and What to Do About Them

Error Ø: NEXT without FOR Error.

This could happen simply like so:
10 FOR I = I To 1Ø
2Ø PRINT I, I x 10
3ØNEXT U

Variable in NEXT statement does not correspond to variable set in the FOR statement.

Another way this could come about is:
10 FOR TU = 1 TO N
5Ø FOR TUNE = 1 TO 8
6Ø P = P + A (TUNE)
7Ø NEXT TUNE

1ØØ NEXT TUT

This part of a programme looks relatively safe, but beware! In APPLESOFT only the first two characters of a variable name are recognised so what we have really done here is:

1Ø FOR TU = ITO N

5Ø FOR TU = I TO 8 : REM RESET TU FOR LOOP
6Ø P = P + A (TU)
7Ø NEXT TU

1ØØ NEXT TU : REM NO 'FOR' HAS NOW BEEN ASSIGNED

So, although we may lose clarity, use variable names, where possible, a maximum of two characters long.

Action to Take:
The programme must be aborted, preferably with the message 'PROGRAMM ERROR AT';
LINEO: END.

Error 16: Syntax Error

This can happen in many wierd and wonderful ways. Here are a few:

10 PRUNT 'HELLO'
U instead of I, simple syntax error
SS IF DI>7 AREA THEN 6ØØ

This is a more tricky syntax error, when this is keyed in, it looks O.K., but when run, a SYNTAX ERROR will appear. The Apple will interpret this line as:

SS IF DI>7 ARE AT HEN6ØØ

hence generating the syntax error. The way around things like this is to put brackets around the offending data name, i.e.

SS IF DI>7 (AREA) THEN 6ØØ

Another type of syntax error is one which involves 'reserved' words. Here are a few nice ones:

```
1Ø TON = CAT
2Ø AGRT = SNEWD
3Ø FOR K = SPLO TO I
```

Look harmless, don't they? But on LISTING, what do we get:

```
1Ø TO N = C AT
2Ø A GR T = S NEW
3Ø FOR K = S PLOT OI
```

Not even brackets can save you from these errors.

Action to Take:
The programme must be aborted again with the message 'Program error at'; LINENO.

Error 22 Return without GOSUB
This error usually occurs when you omit an END statement i.e.

```
 5 INPUT A$
1Ø C$ = A$ : GOSUB 1ØØ
15 INPUT Z$
2Ø C$ = Z$ : GOSUB 1ØØ
25 PRINT 'ANY MORE' ;: INPUT A$
3Ø IF A$ = 'YES' THEN 5

1ØØ PRINT 'LENGTH OF STRING ='; LEN(C$)
11Ø RETURN
```

The programme accepts two strings and uses a subroutine to calculate and print the length of each string. It will work perfectly until you do answer 'YES', then you will generate the error because the programme will fall through to the subroutine and run into the RETURN,

Sample RUN

```
RUN (CR)
? FRED (CR)
LENGTH OF STRING = 4
? ALF (CR)
LENGTH OF STRING = 3
ANY MORE? NO (CR)
LENGTH OF STRING = 3

? RETURN WITHOUT GOSUB ERROR IN 1Ø5
```

By putting line 99 in as so:

```
99 END
```

will cure the problem.

Action to Take:
Again the programme must be aborted as this is a programme error.

Error 42: OUT OF DATA ERROR

The programme is trying to read too many items or is re-reading items that have already been read.

i)

```
1Ø DATA 1, 2, 3, 4
2Ø FOR I = 1 TO 5
3Ø READ A: PRINT A
4Ø NEXT I

REM: ONLY FOUR ITEMS IN DATA STATEMENT
REM: TRYING TO READ FIVE. NAUGHTY!

RUN
1
2
3
4
? OUT OF DATA ERROR IN 3Ø
```

The only way to save this is by altering Line 10 by adding an extra data item.

ii)

```
1Ø DATA 1, 2, 3, 4, 5, 6
2Ø FOR I = 1 TO 6: READ A(I); NEXT I

99 FOR I = 1 TO 6: PRINT A(I): NEXT I
1ØØØ  GO TO 2Ø

RUN
1
2
3
4
5
6
? OUT OF DATA ERROR IN 20
```

The way to save this is either to jump back to a line above the read statement or to use the RESTORE command, to allow the data to be re-read.

Action to Take:
The programme must be aborted as this is a programme error (except for Disc, which is dealt with in the next issue).

Error 53: Illegal Quantity

The official explanation of this is, 'paramenter passed to a math or string function was out of range'. This can happen in many wonderful ways, usually caused by a variable going out of range in a mathematic function.

i.e.

```
PRINT LOG (A)          where A = Ø or AL = 1
PRINT SQR (A)          where A = Ø or AL = 1
PRINT A^B          where A = Ø or AL = 1, or B
                                is not integer.
```

All other functions are prone to this especially string functions, PEEK and POKE functions or graphics functions.

Action to Take:
If you are inputting off the key board then try R:
60020 IF P = 53 THEN PRINT "ILLEGAL" : RESUME
If there is no input then abort run.

Error 77: Out of Memory Error

Like many other errors, this can take place in many ways, such as;
Not enough memory available for program (i.e. loading a 32K program into a 16K machine).
Gosubs nested more than 24 levels deep.
For loops nested more than 10 levels deep.
Parentheses nested more than 36 levels deep.
Setting LOMEN too high or too low, likewise for HIMEM.

An example of this?
LOMEM:500
HIMEM:70000
(If you think I am going to write down 11 FOR loops, etc)

Action to Take:
The program must be aborted

Error 90: Undefined Statement Error.

This error comes about when you try to jump to something that isn't there. That is you have used a GOTO, GOSUB or a THEN statement where you are jumping to a line number that does not exist, i.e:

10 IF A > B THEN 100
20 GOTO 5
30 END
5 INPUT A,B
Here if you entered a value of A which was greater than B, you would receive the above error.

Action to Take:
The program must be aborted as this is a program error

Error 107: Bad Subscript Error

This error happens only with reference to array usage. It comes about when you try to reference an array element which is outside the dimensions of the array. i.e.

10 DIM A(20)
20 INPUT N, A
30 A(N)=A
40 GOTO 10

This program will run happily until you input a value of N which is negative or above 20, then you get the error.

Action to Take:
Obviously in this type of program the range of inputs (in this case N), must be fully validated. By inserting a line

25 IF N < 0 OR N > 20 THEN PRINT "ERROR": GO TO 20

Any other type of occurence should cause program halt.

Error 120: Redimensioned Array Error.

Another array-based error. Arrays are usually given a default dimension of 10, so you would set away by typing

10A(4)=N
without dimensions the array. But if you later on in the program had a line,

400 DIM A(30)
then the error would come about.

Another way this would come about is,
10 DIM A(20)
20 for I = 1 TO 20
30 A(I) = I
40 NEXT I
"
"
"
"
1000 GO TO 10
Dimension statements should be at the beginning of the program.
Action to Take:
Like many other errors the program must be aborted as this is soley a program error.

Error 133: Division by zero error
Basically dividing by zero constitutes an error. This is because if you divide anything by zero the answer is infinity, and the computer does not like infinity. Here is an example,

10 input N
20 input M
30 A = N / M
40 PRINT "ANSWER = ";A
This will work happily as long as M is not zero, if it is then the error will occur.
Action to take:
If the program allows the user to input the offending variables, then make sure the input is validated. In the example above inserting a line

25 IF M = 0 THEN PRINT "MUST BE NONE ZERO": GO TO 20

would save the day. It is doubtful that you would key in a line
10 A = B/0
so normatlly sensible vetting will avoid the error.

Error 163: Type Mismatch Error.

One side of an assignment statement is not the same type as the other. You are assigning a string variable to a numberic variable.

20 A$ = F
or
20 F = A$

or
20 F = MID$(A$,1,2)

In a program if you are inputting a string then it will accept numeric data, but act on it like string data. Ie:-

20 INPUT A$
30 PRINT "STRING IS ";A$
)-RUN
? 45
STRING IS 45
But this is not true the other way around. Ie:

20 INPUT A
30 PRINT "NUMBER IS ";A
)-RUN
? AB

If you have an
ONERR GOTO
line in the program then this will be intercepted with PEEK (222) set to 254, which I will deal with anon. If you do not have the error intercept line then what happens is
?REENTER
?3
NUMBER IS 3
I digress into BAD RESPONCE.

Action to Take.
Normally the program should be aborted as an error of this type is a program error.

Error 176: String Too Long Error.

This normally comes about when you are adding strings together to give another string: Playing around with this I find that the error actually sets PEEK (222) to 254! It can come about like so

10 INPUT A$
20 INPUT B$
30 C$ = A$ + B$
)-RUN
?AAAAAAAA.....Etc
?BBBBBBBB.....Etc
?STRING TOO LONG ERROR IN 30
The maximum length of a string is 255. If you are keying in a string the APPLE starts to beep you as you go over 249 characters and if you persist in entering more it puts a delimiter (/) after 255 and does not accept the string.

Action to Take.
Check the length of strings before you add them. A line,

25 IF LEN(A$)+LEN(B$) ⟩ 255 PRINT "STRING TOO LONG ":GOTO 10

would save the day.

Error 191: Formula Too Complex Error.
This comes about if more than two "IF 'yy' THEN " were executed. i.e.

10 IF A ⟩ B THEN IF B⟩ C THEN IF ...etc

Action to Take.
This is a program error so the program run must be aborted. It should be noted that the program might work OK until A⟩ B and B⟩ C.

Error 224: Undefined Function Error.

Basically you referenced a function which did not exist. i.e.
10 DEF FN A(W) = W - W
20 PRINT FN A (3)
30 PRINT FN B (2)
)-RUN
9

? UNDEFINED FUNCTION ERROR IN 30

Action to Take:
Program must be aborted as this is soley a program error.

Error 254: Bad Response To Input Statement.

There are many, many ways that this can come about. I.e.
20 INPUT C
)-RUN
? fred
or
)-RUN
? (CR) ('RETURN')

Action to Take:
Try something like this,
0 ONERR GOTO 1000
10 INPUT A
"
"
1000 P = PEEK (222)
1010 IF P = 254 THEN PRINT "RE ENTER":RESUMe

Error 255:Control C Interrupt.

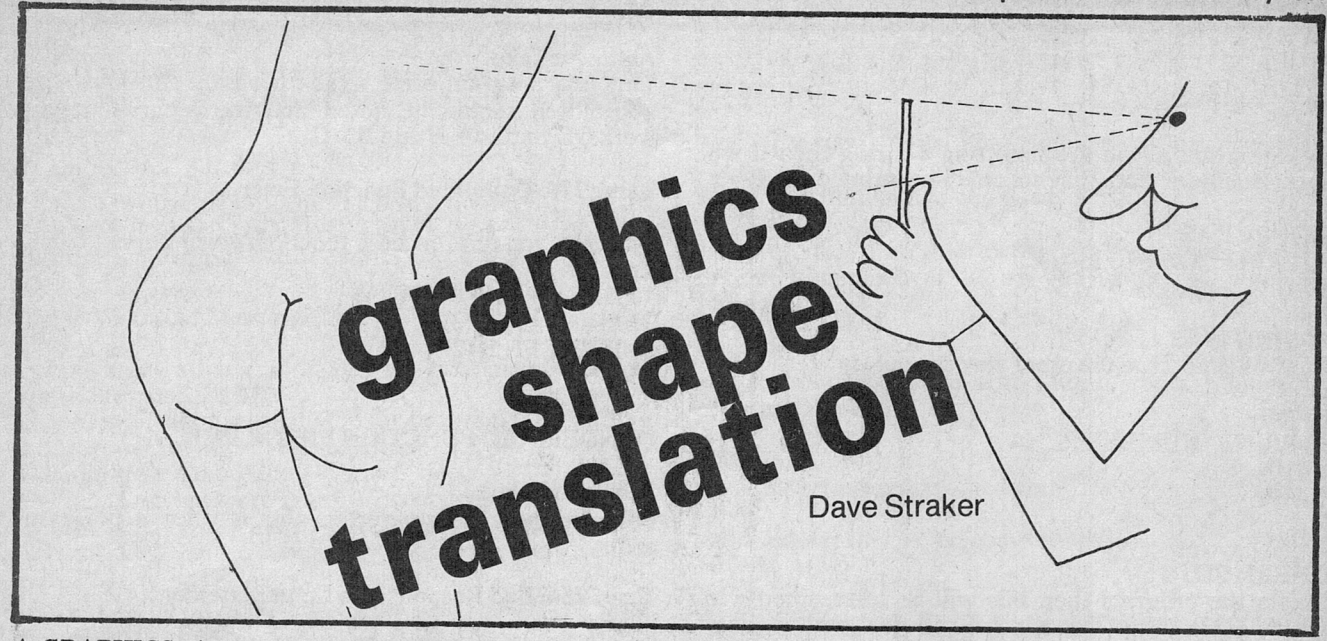
If you don't want people to interrupt with a control C then try 1020 IF P = 225 THEN RESUME

NB. If you hammer the control C all sorts of wonderful things could happen.

This ends the look at the non-disced errors, but there are some things in general to look at ...
1. Take care when handling errors in FOR ... NEXT loops or GOSUBs, you must restart the loop and return to the FOR or the GOSUB.
2. Try to RESUME errors that come about thanks to a bad GET statement with two consecutive bad GETS will hang the system.
3. If you use TRACE or PRINT during a program then you will be leapt into the monitor after 43 errors.

In the APPLESOFT manual on page 82 there is the following routine which might help the above 3 quirks.

1 DATA 104,168,104,166,223,154,72,152,72,96
5 FOR I= 1 TO 10
10 READ a, POKE (767 + 1),A
20 NEXT I
And in the ONERR routine use a
CALL 768
to save the day.



graphics shape translation

Dave Straker

A GRAPHICS shape of a two-dimensional image may be regarded, in terms of translation (i.e. lateral movement) as a point, with all other points (i.e. along lines) keeping the same distance, angle etc. from that point or origin. Thus, to move a shape, we move the origin, and redraw the lines around it, using the shape's fixed drawing routine, (fig.1). Prior to drawing the new image, the old image is deleted.

Thus, a shape is moved in steps, and the two factors that affect the motion are: distance moved per step, and time taken to move that step.

VELOCITY

Velocity is the rate of change of distance. If a point moves through a distance, d, in time, t, then the velocity, v, of the point over that distance is:

$$v = \frac{d}{t}$$

If the point covers equal distances in equal times, then the velocity is cast out (Fig.2). If the point covers different distances in equal times (or equal distances in different times) then the velocity is varying (fig.3).

NOTE: (a) negative velocity means travelling backwards.

(b) zero velocity means not moving.

ACCELERATION

Acceleration is the rate of change of velocity. If a moving point changes from a velocity of v, to a velocity of v_2 in time, then the acceleration, a, over that period is:

$$a = \frac{v_2 - v}{t}$$

If over equal periods of time, the velocity changes by equal amounts, then the acceleration is varying (Fig.5)

NOTE: (a) negative acceleration means de-acceleration (retardation).

(b) constant velocity means zero acceleration.

THE EQUATIONS OF MOTION

There are a set of equations that may be used to calculate any one of: acceleration, initial velocity, final velocity, distance and time over any period, given any three of the others (Fig.6)

EXAMPLE: If the velocity of a point at A is 5, and it moves with acceleration 2 directly to point B in time 3, then:

$u = 5$

$a = 2$

$t = 3$

$s = ?$

$= ut + \frac{1}{2}at^2$
$= 5 \times 3 + \frac{1}{2} \times 2 \times 3^2$
$= 24$ units from A to B

TWO DIMENSIONAL MOTION

Motion in multi-dimensions may be achieved by considering the motion in each dimension separately. We may look upon the area of movement as a system of cartesian co-ordinates (Fig 7). This usually convenient for computer reference also. All plotting is to be done relative to the origin of the frame of reference (i.e. the screen) rather than calculating points from the previous point, in order to avoid propagating errors. The motion of a shape may be determined by external input such as keyed signals, predetermined laws, or a combination of the two.

As two dimensions are being used, then two values must be used to determine the position of any point. These may be x and y (cartesian) or r and θ (polar). Given one set we may change to the other (Fig.7) such as changing the angular motion of moving in a circle to cartessian values to plot on the screen.

MOTION UNDER GRAVITY

There are two basic rules for plotting motion of a point under the effect of gravity:

(a) The only acceleration is vertically downwards and is constant, $g(= 9.8 m/sec^2)$

(b) Use the equations of motion.

The normal situation is that the point is given an initial velocity at a set angle to the ground (Fig.8).

There is no other acceleration other than that due to gravity, because an acceleration other then that due to gravity, because an acceleration is produced by a force, and there is no other force when the point is moving freely, after the force applied to throw it is removed.
EXAMPLE: Given an initial velocity of u at an angle θ, to find x and y, the co-ordinates of the point after time, t
Using Fig.6 equations and Fig.8
To find x, consider the horizontal direction only.
$u = u \cos \theta$
$a = 0$
$t = t$
$s = x$

$x = ut \cos \theta$

to find y, consider the vertical direction (up is positive) only.

$u = u \sin \theta$
$a = -g$
$t = t$
$s = y$

therefore $y = ut \sin 0 - \frac{1}{2}gt^2$

the time to landing is found by making $y = 0$ in the last equation:

$$t = \frac{2 \sin 0}{g}$$

hence to plot the movement of the point, movement t from 0 to 2 sin 0/g, calculating x and y each time.
NOTE: the path travelled is a parabola.

ANGULAR MOTION

The laws for notation, such as in a circle, depend not upon the change of x or y, but 0, the angle turned. The rate of notation is called the angular velocity, w, such that over a given period, t, if the angle 0 is turned through, then the angular velocity over that period is given by

$$w = \frac{\theta}{t}$$

hence the co-ordinates of the point moving in a circle may be given by (form Fig.9)

$x = r \cos wt$
$y = r \sin wt$

NOTE: Positive angular velocity gives anti clock wise motion, negative angular velocity gives clockwise motion, constant angular velocity gives a steady rotation rate.
Other rotary motions may be similarly described, such as movement through an ellipse (Fig.10)

COMBINING MOVEMENTS

Often several influences simultaneously affect the motion of a point. To resolve the overall effect, the displacements caused by each influence must separately be calculated, and the co-ordinates summed.
EXAMPLE: Consider a planet, travelling in a straight line, being orbited by a moon (Fig.11). The prolate cycloid described by the moon may easily be calculated:
Movement of planet relative to absolute zero:
$y_1 = y_1$ (constant)

$x_1 = vt$

movement of moon relative to planet:

$y_2 = (h+r) \sin wt$

$x_2 = (h+r) \cos wt$

therefore movement of moon relative to absolute zero:

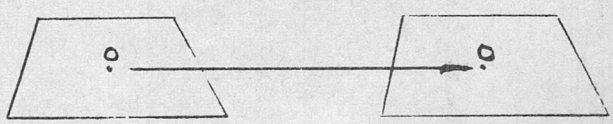
$y = y_1 + (h+r) \sin wt$

$x = vt + (h+r) \cos wt$

SOME PRACTICAL CONSIDERATIONS

To move a shape at a given velocity, the main prlblem is to give an image of smooth, not jerky, movement To accomplish this, the shape must be redrawn in a new position as often as possible. The optimum rate for this is about 25 per second, giving you 40 ms to do your calculation, delete the old shape, and draw the new. To minimize time:
(A) don't re-calculate constant values.
(B) keep unnecessary junk out of your movement loop.
(C) use machine code(!)
Another point to watch is not to let your code try to draw part of the shape off the screen!

Fig1.



Origin moves, shape follows

Fig 2.

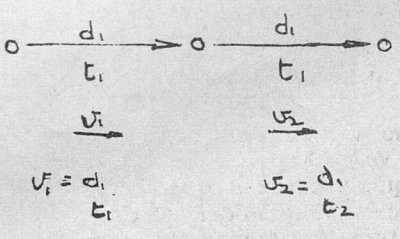
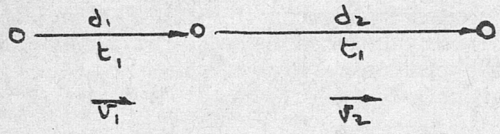


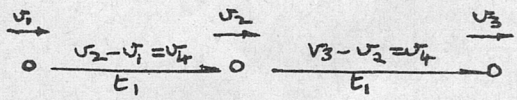
Fig 3.



$v_1 = \frac{d_1}{t_1}$ $v_2 = \frac{d_2}{t_1}$

$v_1 \neq v_2$, varying velocity

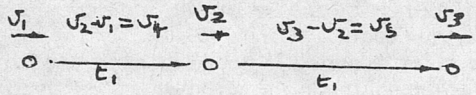
Fig 4



$a_1 = \frac{v_4}{t_1}$ $a_2 = \frac{v_4}{t_1}$

$a_1 = a_2$, constant acceleration

Fig 5.

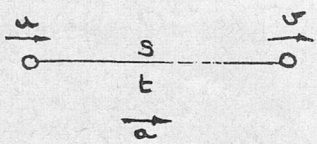


$a_1 = \frac{v_4}{t_1}$ $a_2 = \frac{v_5}{t_1}$

$a_1 \neq a_2$, varying acceleration

Fig 6.

diagram:



symbols:
- u = initial velocity
- v = final velocity
- a = acceleration
- s = distance
- t = time

equations:

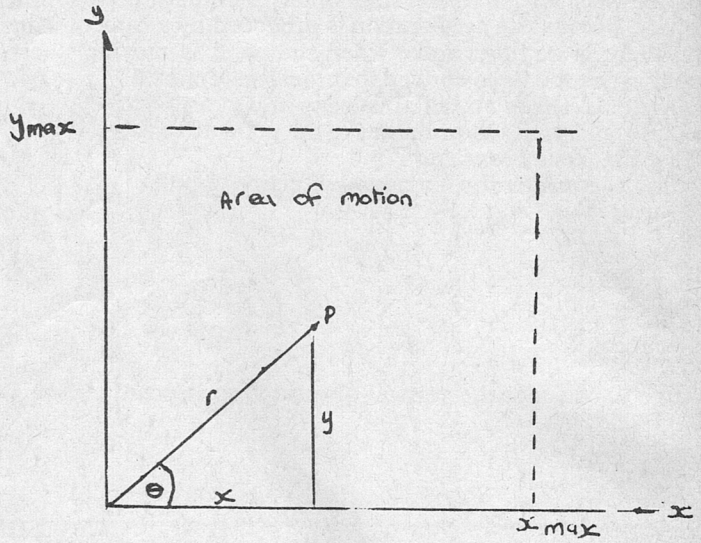
$v = u + at$

$s = \frac{1}{2}(u+v)t$

$s = ut + \frac{1}{2}at^2$

$v^2 = u^2 + 2as$

Fig 7.



polar → cartesian

$x = r\cos\theta$

$y = r\sin\theta$

cartesian → polar

$\theta = \tan^{-1}\frac{y}{x}$

$r = \sqrt{x^2 + y^2}$

Fig 8.

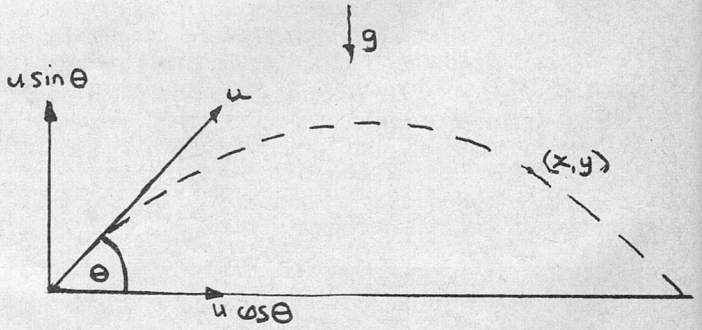
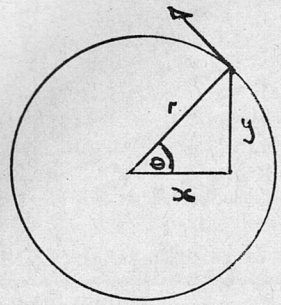


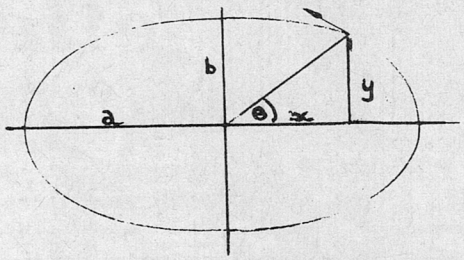
Fig 9.



$x = r \cos\theta$

$y = r \sin\theta$

Fig 10

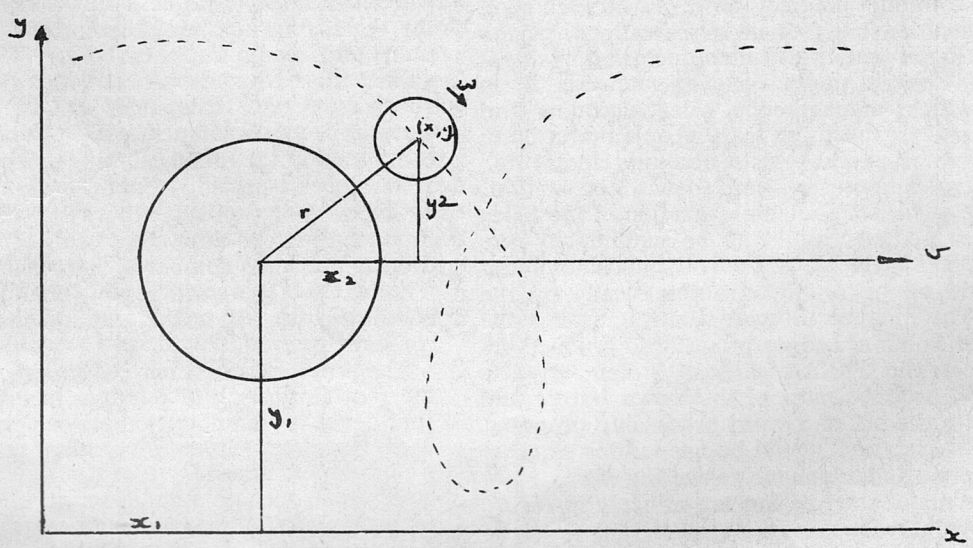


$x = a \cos\theta$

$y = b \sin\theta$

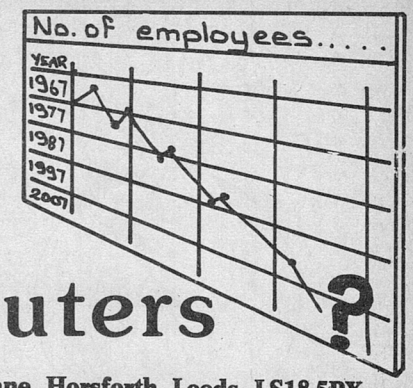
$\theta = \omega t$

Fig 11



planet, radius, r, moves with velocity v.
moon orbits at height, h, with angular velocity, w

The social effects of micro computers



P R RAYNER,
B.A., M.B.C.S., Great Northern Computer Services Ltd., 116 Low Lane, Horsforth, Leeds, LS18 5PX.

†Until a few short years ago computers were to be found only in large organisations. The development of commercial micro-computer systems has changed all this. For about £4,500 a small businessman can obtain a complete micro-computer system with Visual Display Unit, Printer and software for accounting and word processing. When leased over three years, such a system will cost about £170 per month; much less than the cost of a clerk or a typist.

Yet this modern electronic marvel can do more than any one clerk or typist. During the course of the month it can keep to stock records up to date, print the invoices and statements, produce the Balance Sheet and Profit and Loss Statement and help the sales department to produce inumerable letters, proposals and reports.

The micro computer is making an impact for reasons other than cost. The equipment fits neatly in the corner of an office. The programs are self explanatory and easy to use. The result is that no computer staff are needed to run the system. Thus, for example, the stores controller updates his own stock records instead of having a separate clerk to do the work.

The micro computer may not, however, be such good news for school leavers with no qualifications, particularly girls. In the past a small manufacturing business employing 40 people might have as many as 20 in indirect activities; management, sales, accounts and administration. Of these, up to 10 people might be in clerical or secretarial jobs; typists, accounts clerks, etc.

With a micro-computer system as few a 3 or 4 might, in theory, be needed. The administration of the Sales and Purchase Ledgers would all be handled by one accounts clerk. Invoices, Statements and Reminder Letters would all be produced automatically by the computer. The Finance Director would himself make entries to the Nominal Ledger program to produce the Balance Sheet and Profit and Loss Statement. The salesman will produce most of thier own letters and proposals with the aid of a word processing program. Only 2 or 3 'secretaries' would be needed for all other typing, reception and telephone answering work.

In practice major staff saving are unlikely to result. There are several reasons for this:

Firstly, the computer can produce much more infromation. For example, regular weekly accounts may reveal situations that had not previously been noticed and these situations must be investigated. Thus, by producing more information, the computer produces new tasks for staff. The end result is a more profitable business, but not necessarily a reduction of staff.

Secondly, there are few staff in a small company who have no 'secondary' tasks, even though is a manual system they usually only have time for 'essentials'. These essentials are often the routine, mechanical parts of office life which are most suitable for computerisation. When they are computerised, the staff have time for the secondary tasks. In the Accounts Department, for example, the Sales Ledger clerk can now spend his time telephoning defaulting payers. Meanwhile the Purchase Ledger clerk can be negotiating better discounts with suppliers. Again, the result is a more profitable business but no loss of jobs.

Thirdly, growth will soon absorb any 'spare time' that results from computerisation. Computers and growth seem to go together. It is unclear as to whether organisations grow because they are continually installing better methods, including computers. Whatever the relationship a computerised office means that a business can grow successfully without increasing office staff.

Forthly, computers create new jobs. Programmers and analists are needed to prepare the software, engineers are required to maintain and repair the hardware; salesmen, designers and workers are needed in the increasing number of factories when micro-computers and media supplies. Few of the jobs will be within the end-users office. Mostly they will be in specialist service companies and in the factories and workshops of manufacturers. None-the-less, micro computers in the office can only work with the support of a large and growing force of 'back-up' workers.

Thus, it is unlikely that the advent of 'mass-produced' micro-computers will cause a significant drop in the total number of jobs available. However, it is clear that there will be a big change in the character of office jobs. It is this change that is a cause for concern.

The problem is that not every person who is good at the routine posting of a ledger has the personality or confidence to 'dunn' customers or negotiate with suppliers: Not every secretary is willing or able to become a programmer; not every typist who is content to transcribe other people's manuscript into typed copy is happy to work with a word-processing system. In the new age of the automated office such staff are at a disadvantage. In such an office the routine work is done by the computer, whilst people are required to provide the initiative, drive and decision making that no machine can give.

What will happen to staff who are unable or unwilling to exhibit the necessary drive and initiative! Some of the women may decide to stay at home as full time

wives and mothers. Others may go to fill the many vacancies that currently exist in industries like public transport. Perhaps domestic service may make a comeback.

Without a crystal ball it is hard to predict. One can only say that 'if present trends continue and all else stays the same then.....!' Of course every thing else does not stay the same and the dire forecasts of sociologists and economists are rarely born out.

It is clear, however, that some jobs will be destroyed, new jobs will be created and many jobs will change in character. It is also clear that education at several levels will be the key to overcoming these changes.

At the most basic level it is obvious that our education system should gird itself to install more drive, imagination and intitiative into school leavers.

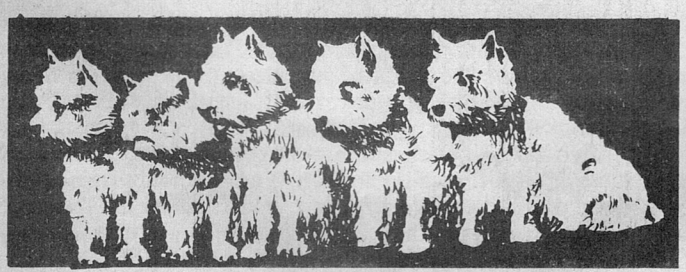
At a more advanced level we must ensure that every school has access to a computer and every school leaver has a basic understanding of what micro computers can do. In addition computer programming should be made as widely available as mechanical drawing or commercial studies as a training for the programmers of the future.

Facilities must also be provided to retrain mature office workers whose jobs have changed.

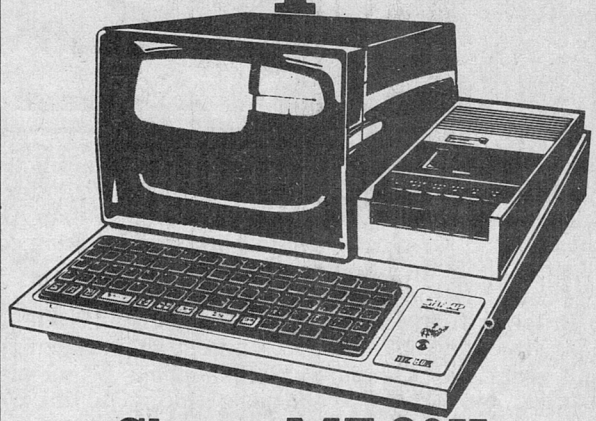
Given the will, these major developments could take place even in the current financially straightened times. Industry and commerce are already suffering from a shortage of skilled programmers and experience shows that local firms are often ready to help to provide computer facilities. Tax concessions to encourage donations would further increase the number of firms ready to help in this way. The relatively low cost of a small micro-computer system is well within the scope of a well organised Parent Teacher Association.

Teaching staff with the right experience and attitudes are probably harder to organise. Here the micro computer itself can help by being both the subject and the teacher. The PILOT program as featured in this magazine, is one way of doing this.

The resources to provide this major initiative in education are available now. What may not be available is the will to cut through ignorance, incompetence, bureaucracy and 'red tape' to bring them together. If, as a nation, we fail to rise to this 'challenge of the micro' then the twin evils - unemployment and staff shortage will blight over lives in the '80s and '90s even more than they have in the '70s. If, on the other hand, we do rise and meet the challenge, then the micro-computer can herald in a new, more productive, brighter and better Britain.



Dare you compare!



Sharp MZ 80K

NOT A KIT
Works the day you buy it

JAPANESE
The same quality they have put into cars and Hi-Fi

SINGLE UNIT
No trailing leads and wires

Z80
More registers and instructions than other processors.

TAPE BASIC
You don't get left with obsolete ROMs.

TAPE COUNTER
Know where you are on the tape.

SOUND
Built-in music synthesiser with 3 octaves.

FAST LOADING
Cassette interface runs at 1200 bps.

Other features — 79 keyboard, up to 48K RAM, on screen editing, real time clock 256 different characters, 10 inch video display 80 x 50 bit mapped graphics.

DISK DRIVE AND PRINTER COMING SOON

Memory after loading 14K Basic	Nett	VAT	Total
6K	520.00	78.00	598.00
10K	540.00	81.00	621.00
18K	620.00	93.00	713.00
22K	640.00	96.00	736.00
34K	740.00	111.00	851.00

All prices include courier delivery within UK.
Access and Barclaycard Welcome.

MICRODIGITAL

25 Brunswick Street, Liverpool L2 0BJ
Tel: 051-236 0707 (24 hour Mail Order)
051-227 2535 (All other Depts.)
Mail orders to: MICRODIGITAL LIMITED,
FREEPOST (No Stamp Required) Liverpool L2 2AB.

Futuresoft

by

JULIAN ALLASON

FROM humble beginnings in a Berkshire farmhouse, Petsoft has grown into Europe's largest distributor of microcomputer software. Here its founder, Julian Allason, looks at the recent past to draw some striking conclusions about the future of micro-software.

Britain's first computer store opened in a Fulham back street and a surge of excitement all of three and a half years ago. Computer Workshop was a tiny place manned by teenage enthusiasts selling South West Tech equipment at robust prices. The brochures were unreadable save by other enthusiasts. Their number can be gauged pretty accurately since almost all belonged to the Amateur Computer Club; membership in 1977 was less than 300.

That year, I went to America to investigate a beguiling rumour with the initials P.E.T. It was packaged as a single box, had BASIC resident and was pretty, withal.

American computer stores were packed with equipment: Imsai, Sol, Sphere; none of these are still made, many of the original manufacturers having gone out of business. Forget the PET they said, buy a real computer. Eventually, in someone's back office, I was shown a PET. It was love at first sight. Any plans I may have nurtured of building a British microcomputer vanished that instant.

Surprisingly none of the computer stores we visited had any real applications software to sell. This set me thinking. After all, we in Britain were supposed to be better at software than anyone else.

In March 1978 my PET arrived; serial number 000008. Despite the assistance of a mainframe time-sharing manual prepared by two illiterates from the NCC, I had managed to master the rudiments of BASIC. PETSOFT WAS BORN IN OUR KITCHEN.

At first we published games, then an Assembler/Editor, various utilities and our first business programs. The company grew rapidly, doubling its turnover three months running. We had also begun to publish programs submitted by other PET users.

Originally we had supplied by mail order, but now the growing number of PET dealers were demanding top class business systems, and that meant seeking the help of a large computer company.

Looking at the leading software houses, I concluded that the best placed to help us was Applied Computer Techniques Ltd., then and now Britain's most successful computer services group.

Almost immediately after the takeover ACT began to pour money and resources into developing a range of business packages for the PET which once completed could be linked via database to form a truly integrated suite of business programs. By Easter 1980 we had sold more than 100,000 programs.

Others followed suit and there can be little doubt that Britain now leads the world in micro software. Given the rapid changes in the type and quality of micro software during the past three years, most experts are reluctant to predict future developments.

Nonetheless here is what will happen: more and more dealers and small software houses will develop stand-along applications and packages. The best of these will be snapped up by the large commercial distributors for publication worldwide. The dealers, noting the cost of advertising, distribution, and software maintenance, will tend to standardise on a few well known packages plus programs of their own devising. This pattern has already begun to establish itself. The bulk of the market will go to a number of big distributors like Lifeboat Associates who will take on more and more the character of publishers and wholesalers. Since support from such monolithic organisations will be limited, documentation is likely to improve dramatically.

Individual users will themselves develop packages which through sheer quality will establish a permanent position for themselves in the market-place. Michael Shrayfer has already succeeded in doing this in America with his Electric Pencil word processing package, which is now available for over seventy different computers. Pete Dowson is well on the way to a similar success with his Wordcraft W.P. package here in Britain.

A serious outbreak of piracy, such as is already occurring in Germany and Holland, will be suppressed by the sheer cost of litigation. The large software houses won't care if they lose at the end of the day. Such a case could easily cost £50,000 just to defend.

The immediate future offers a wealth of low-cost database management systems. Their availability will in itself fuel the sales of microcomputers, since databases enable the computer to function in the way the user-in-the-street expects them to.

Two major developments on the hardware front will have a significant impact on type of commercial software available. The first is the arrival of inexpensive accoustic couplers and modems. These will allow microcomputer users to communicate via the GPO telephone network, swapping software and messages. Electronic mail is already a reality in the United States, via several

large micro-networks.

The GPO bureaucracy has been thoroughly shaken by the loss of their monopoly on equipment to be connected to the telephone network. Post Office approval of accoustic couplers is already being speeded up and it should be possible to buy one for less than £100 by the end of the year.

Several of the major software suppliers, including my own company have been examining the feasability of supplying programs over the telephone. The way it will work is as follows. By calling a special number your micro is connected to our ICL2900 series mainframe. Once online you can access detailed and up-to-date information on any software package in our library. To order, you merely key-in the details of your credit card number, address and the program you have selected. It is then down loaded over the telephone to your computer for storage on disk or tape, and your credit card account is debited accordingly. Lest anyone doubt the practicality of this, MicroNet are already offering telesoftware to North American subscribers.

A qualitive change in applications software will result from the availability of hard disk devices of between 10 and 125 megabytes of store. Several such mini-Winchester devices are already on sale and the price is falling.

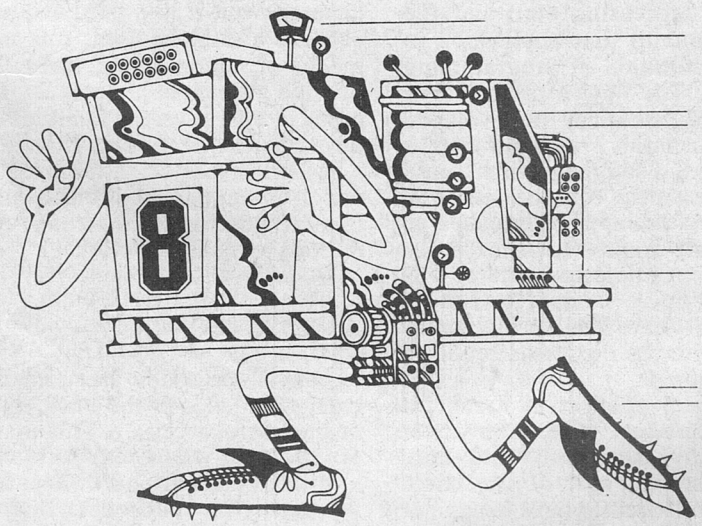
Since the average novel fits comfortably into one megabyte of memory, it will be seen that the information handling capabilities of the microcomputer will have been transformed at a stroke. The prospect of the electronic encyclopedia is not an unrealistic one. Pity only the poor soul who has to key it all in.

It seems likely that in the future technical journals, railway timetables, various other reference works and indices, possibly even fiction, will be published in the form of magnetic media for transfer to hard disk storage devices. Information on Wall Street stock movements is already available on subscription for $175 a quarter.

Much of what we now regard as software, both of the systems and applications variety, will appear in solid state form in the future. One of the first plug-in firmware packages was the Programmers Toolkit, a series of additional commands for PET's BASIC. Despite a £55 price tag it was hugely successful.

Several companies, notably Transam already supply the systems software for their computers on EPROMS - eraseable programmable read only memory chips. To upgrade from BASIC to Pascal, the EPROMS are wiped and the new language and operating system burned in.

Now that some of the business packages have been around long enough for the bugs to be connected, there is no reason why they too should not be ROMed. Well, actually there is one reason: since the architecture of most 8-bit micros allows only 64K of memory to be addressed (without bank switching), not much RAM is left after the business ROMs have been plugged in. However, with 16-bit micros, it is rather different. But that is another story.



Programming Practices and Technics

MICROCOMPUTER PROGRAMMING TECHNIQUES

PROGRAMMING LANGUAGE

Part IV

Martin D. Beer,
Computer Laboratory
University of Liverpool.
P.O. Box 147
Liverpool,
L69 3BX

THERE has been some interest in recent months in the various languages used to program microcomputers. The usual approach has been to compare how short program segments can be coded in the different languages. Whilst this can be very valuable in the study of design, and implementation of languages, and in the comparision of those which are intended for similar uses, it does little to illustrate which languages are most suitable for use in a particular application. In this article I hope to show how, by using an appropriate language for the task in hand, the programmer's job can be eased considerably. Most languages have been designed with a particular use in mind. (BASIC and PASCAL for teaching, ALSGOL 68 for numberical programming, LISP for list processing etc.) and the facilities provided are chosen with that application in mind. At first sight the proliferation of programming languages is very confusing. The novice programmer often asks why just one language cannot be developed which includes all the useful facilities provided by other languages. Unfortunately such a language, if it could be developed, would be extremely complicated, and difficult to use. The compiler, or interpreter, used to translate it into executable code would also be far larger than could be fitted into current micro computers. It is therefore necessary for the micro computer user to select a number of languages which provide the facilities that he will require to write the programs required by the applications he has in mind.

Figure 1 gives some typical application areas for micro computer programs. The range is vast, far larger in fact than that covered by mainframe and mini computer programs since the low cost of the micro computer allow it to be used in areas where it has not been economically viable to use computers before. At one end it is essential to be able to tell the processor exactly what you want it to do without any ambiguity. This requires an intimate knowledge of the components which make up your system, and how they interact with each other. The programs are usually fairly simple so that the programmer is able to convert his problem into a form which is aminable to coding as a series of machine instructions. At the other end of the scale the programmer wishes to write down his problem in the way that he thinks about it, and is not concerned about the underlying structure of the machine which he uses to help him solve it. He does not want to know about the intricasies of handling floating-point arithmetic, or matrix manipulation, these and many other similar problems can be handled quite adequately by the computer. In between there are a whole range of applications in which some knowledge of the way that the underlying hardware operates, but the programmer can leave much of the mundane organisation to the computer.

If the application to be programmed involves considerable numerical computation then the provision of a wide range of floating-point operations is an absolute necessity but if it is intended primarily to collect, consolidate and store data, this is not together with the means of manipulating them. Whilst most languages provide at least some means of performing both these types of operation the facilities offered are more highly developed in some areas than in others.

It is therefore up to the programmer to select a collection of languages to run on his micro computer which is most suitable for the range of activities in which he intends to make use of it.

ASSEMBLER

Assembler codes are converted with a one-to-one correspondence to the basic machine code instructions of the microprocessor. This makes it a very useful tool for the man who needs total control of the processor, and its supporting hardware. Since all programs must eventually be reduced to their machine code instructions, in theory, all application programs can be written in Assembler. In practice however this is rarely done since using Assembler ties you to a particular processor (and indeed, very often to one single machine) and because the programmer has to handle everything is very expensive both to write and maintain. Assembler programs are "Machine Orientated" in that primary coding considerations are concerned with the activities of the processor and its supporting hardware, not the problem which the programmer is attempting to solve.

BASIC

BASIC is by far the most popular language for programming microcomputers. Having been designed as a primary teaching language it is very easy to learn, and use and being particularly suitable for implementation through an interpreter is comparatively easy to instal on a microcomputer system. The program is written in a form which is very similar to that used by a programable calculator and so is already familiar to many newcomers to computing.

BASIC is most suitable for writing computational short programs. Because of the simplicity of the language and the ease of debugging when using an interpreter sort of program can be developed very rapidly. Only when an attempt is made to develop longer, and more complex programs do the limitations of the language appear. The most obvious are the lack of a proper procedure calling mechanism, which allows the programmer to pass essential information to the procedure in a clear, unambiguous form, and the very restricted range of data structures available. Both these can be countered, to a large extent, by adopting strict coding rules, which must be developed to handle particular applications, but this throws a considerable extra burden on the programmer, which he will probably be willing to carry.

Because BASIC has been available for some time on a wide range of microcomputers it is the most prevalent language for distributing software. Unfortunately this means that it has been used for a whole range of applications for which it is not really suited. (It has even been used to write a compiler). This has led to a concerted campaign against BASIC in the microcomputing press. However it should not detract us from the fact that BASIC is a very useful, and if used properly, powerful programming tool, which hopefully will be with us for a long time to come.

PASCAL

The language which is taking over from BASIC, for writing large microcomputer programs is PASCAL. PASCAL was developed by Wirth at ETH, Zurich primarily as a language to teach students how to program. It is a much more flexible language than BASIC in that variable names may be restricted, and statements are not restricted to one line. The basic data types can be nested almost indefinately so that data structures can be defined which accurately represent the needs of the problem. PASCAL also has a proper» procedure handling capability which allows libraries of useful routines to be developed and used in different programs.

Hardware-control functions can often be expressed directly with PASCAL statements, or with small Assembler code procedures which are linked to the PASCAL program in the same way as library procedures. This adds greatly to the power and flexibility of the language when writing "process control type" programs. The wide range of control structures allow the program to be designed and laid out in a very clear and logical way.

As with all programming languages PASCAL has its annoying points. Three which come immediately to mind are the inability to pass an array of any length as a parameter to a procedure, the unexplained lack of an exponentiation operator and the lack of an exception clause to the CASE statement. The first and last of these are fairly easy to program round, the second is a real handicap in developing user libraries.

APL

APL is a very powerful language specifically designed to make the manipulation of vectors and matrices very easy. In suitable applications programs which would take hundreds of lines in any other programming language, take just a few lines in APL. Its supporters stress this when stating the case for its value as a general purpose language. Unfortunately it is very easy to write cryptic and unreadable programs, which do not express underlying data and control structures which were used to develop the program. This is a considerable disadvantage, particularly when the program is unrelated to array handling, when the advantages of brevity do not apply. It is therefore probably best to consider APL as a special programming language which is particularly useful for applications which require many complex matrix operations.

LISP

LISP was designed by John McCarthey at MIT during the early 1960's. It is different from most other languages considered here in that it is not algorithmic. It may be calagorised as a functional language, as a symbolic language, as a list of processing language, as a recursive language and as a logical language.

Being a functional language every type of construction which may be performed in LISP, and which can be defined as a function in the mathematical sense is written as a function in LISP. Thus if we wish to add two numbers X and Y we would write

(PLUS X Y)

(This is in fact exactly equivalent to the handling of operators in other languages, since both unary and binary operators can always be replaced by a suitable function).

One of the most important features of LISP is that the data which it manipulates need not necessarily be composed of numbers. There is therefore no assignment statement, as such. Another special function is used instead e.g. to set the contents of a variable named X to 3 the following code would be required.

(SETQ X 3)

Programs are built up by defining new functions which use other functions, which have already been defined. These operate on the data given to them to provide the final solution to the problem.

PILOT

PILOT was developed to assist teachers in the development of Computer Aided Instruction programs. It was intended that it should be as simple as possible to learn and use, and should be free of assumptions about the particular learning philosophies to be used.

PILOT is built around a very limited series of in-

structions (eight in "core PILOT" and thirteen in "common PILOT") which perform all the commonly used operations in Computer Aided Instruction programs. These are programmed as single letter operation codes which are easy for non-specialist programmers to remember and use.

PILOT is really an extremely comprehensive applications package for writing CAI programs. As such it removes much of the necessity for the teacher to know about the intricasies of programming. Using the PILOT interpreter he is able to evolve up CAI sessions to suit his own style of teaching.

Conclusions

Hopefully this article explains why so many different languages are used to program microcomputers. As more application areas are developed it is likely that special programs will become available which will remove the programming requirement, at least in the traditional sense. A good example of this is VISICALC, which is effectively a computer-maintained worksheet. The user configures it to his requirements, and the system is then ready for use. Packages like this will become prevalent over the next few years, and will ease the microcomputer user's lot considerably.

Further Reading

D W Barron
"An Introduction to the Study of Programming Languages"
Cambridge University Press

D W Barron
"Assemblers and Loaders"
Macdonald/James
W D Maurer
"The Programmer's Introduction to LISP"
Macdonald/James

I R Wilson and A M Addyman
"A Practical Introduction to PASCAL"
Macmillan

A Learner and A J Powell
"An Introduction to Algol 69 through Problems"
Macmillan

A J T Colin
"Programming and Problem Solving using ALGOL 68"
Macmillan

B Kerninghan and P Plaugher
"Software Tools"
Addison-Wesley

P Henderson
"Functional Programming, Application and Implementation"
Prentice-Hall

Common PILOT Manual

D Straker
"I'm PILOT, Fly Me"
Liverpool Software Gazette No.1 page 28

K Chung and H Yuen
"A Tiny PASCAL Compiler"
Byte September - November 1978.

A very useful introduction to the UCSD PASCAL system
Beginner's Guide for a UCSD PASCAL System by Kenneth Bowles (Byte Books)

Compliments Microcomputer Problem Solving using PASCAL by the same author in that it deals almost exclusively with using the UCSD operating system. There are chapters describing the use of the Filer and Editor, the diagnostic information generated when either a compilation or an execution error is detected, and on the use of files and user libraries. Everything is explained far more clearly than it is in the PASCAL Reference Manual.

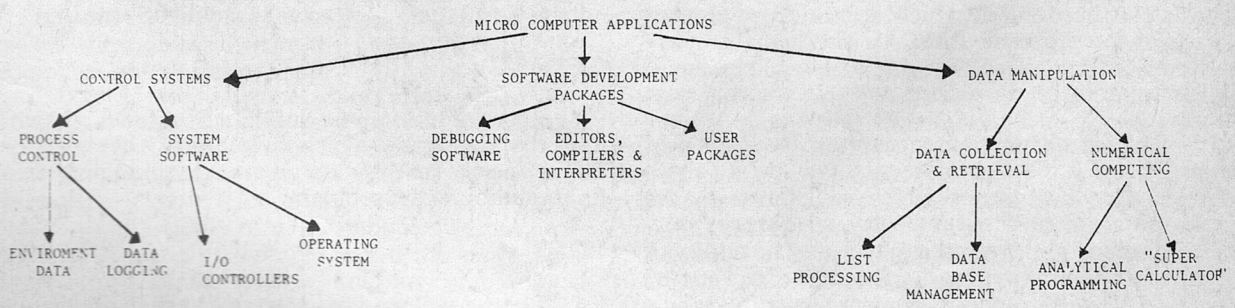


Figure 1

Some of the wide variety of applications for which micro computers are programmed. To the left the programmer requires a language which gives him access directly to the facilities offered by the micro computer he is using. To the right he does not require this, but with the increased complexity of his programs requires a language which is problem orientated.

ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC
ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC
ETC ETC
ETC ETC
ETC ETC

ETCETERA

ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC
ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC
ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC

PET SPEAKS

PETSOFT have released a 'Talking Calculator' program which generates synthetic speech via an inexpensive soundbox connected to the user port of the Commodore PET microcomputer.

The £10 program is thought to be the least expensive piece of speech synthesis software yet to reach the market.

Once loaded the program causes the name of each numeric or function key to be generated as synthetic speech, followed by the result of the calculation. The representation of a calculator on screen allows the calculation to be tracked visually as well.

Problems are entered using the numeric keypad. To divide six by four, the '6', '/', '4' and '=' keys are pressed. As each key is depressed the computer responds with the words: "Six ... divided by ... four ... equals ... one ... point ... five".

Petsoft have already received a number of enquiries from the blind. The program also enables calculations to be made without taking the eyes off a column of figures. Errors due to miskeying calculations were found to be much reduced as a result of audible key identifation. The company plan to release more programs incorporating speech synthesis in the near future.

The 'Talking Calculator' is available price £10+VAT from PET dealers or by mail order direct from ACT Petsoft, 66-68 Hagley Road, Edgbaston, Birmingham B16 8PF. Telephone 021-455 8585. A User Port Speech and Music Generator with volume control and connectors is also on sale priced at £27 + VAT.

PHILIPS FORMS NEW £100 MILLION GROUP FOR THE ELECTRONIC OFFICE MARKET

PHILIPS Industries has integrated four operating companies to produce a single group - Philips Business Systems - aimed at supplying the total needs of the electronic office.

In its first year - 1980 - the new Group's turnover will exceed £100 million. By 1984 this figure will have doubled to more than £200 million, giving Philips a 22 per cent stake in the fastest-growing electronics market in the UK.

The four companies brought together under Mr Brian Manley, Managing Director of Philips Business Systems, are: Pye TMC Ltd, one of the world's leading innovators in telephone equipment; Philips Data Systems, originators of the "Office Computer" and strongly positioned in both business computer and terminal markets; Pye Business Communications Ltd., specialist suppliers of intercoms, internal communications systems and computer controlled telephone switching systems; and Philips Business Equipment, who hold over 65 per cent of Britain's word-input market and supply the highly-regarded P5002 word processor.

According to Brian Manley, the new Group has been formed to exploit Philips' unique position in the electronic business equipment market of today: "Philips has the broadest range of products in Europe - for the electronic office", he points out. "We have drawn together our strengths in manufacturing, marketing, systems engineering and support in order to make a powerful, unified attack on a market which is both expanding and converging".

Philips predict that the fastest-growing sector of the business systems industry in the next decade will be in communications-based equipment, which is where Philips regard themselves as strongest. In fact, Brian Manley considers that telecommunications will form the basis of the electronic office, where intellegent, personal terminal units will analyse, retrieve from, and contribute to a high-speed digital data-stream passing through the work location.

According to Philips' assessments, the rapidly-growing market for electronic business equipment will evolve in two distinct phases. For several years we will see an increase in the volume of "stand-alone" equipment installed; more and more, these will possess their own intelligent communications capability. Phase Two will involve the integration of equipments themselves, until, during the 1990s complete inter-communication facilities are achieved amongst terminals handling word and data processing, audio and message transmission, data and text storage, and a wide variety of "personal computing" functions.

Announcing the new Group, Brian Manley pointed out that the major growth in this market will be based more and more upon computing power at the desk; the role of the large mainframe computer will become more specialised, and the market is virtually saturated. Philips is strong in the technologies of small computers, distributed processing, and intelligent communications systems, and is well placed to exploit this growth.

The new Business Systems Group is to be launched with a major advertising campaign in the national press, and will initially be based in a new headquarters in Maidenhead. The process of integration will continue throughout the next two to three years, at first in the areas of product planning, marketing and engineering support.

Brian Manley forecasts that more than half of the new Group's turnover will come from products which it makes in Britain. The telecommunications activity of Philips Business Systems has manufacturing facilities at Malmesbury and Airdrie; substantial growth is expected in this sector, as a result of new product development and the increasing role of telecommunications in the office.

Philips is unusual amongst equipment manufacturers in having almost equal interests in the professional and consumer markets. Brian Manley pointed out that convergence is drawing these traditionally separate areas closer together, so that new products - the Videodisc, for example - can have far-reaching applications in both office and the home. This dependence on both market areas also encourages Philips to adopt a balanced view of the impact of electronics on the economy, one result of which is Philips Business Systems' investment in a major education and training programme designed to help people understand innovation and exploit change.

Discussing Philips' long-term intentions in the business systems market, Brian Manley referred to the confusion which characterises the industry and its future when seen from the viewpoint of the customer. "Electronic business equipment and office electronics seem a "jigsaw puzzle" to even the best-informed observers of the market", he said. "What we have done is to identify the pattern of future developments, take the appropriate pieces and put them together to form a coherent picture. Our aim is to introduce new technology to the market in a way that will enable customers to genuinely benefit from it - without trauma, without risk, and in complete assurance of our continuing support. Above all, what we need to do is to convince British business that the so-called "office of the future" is here today, and that industry and commerce most invest now to make their business operations more effective."

APPLEWARE OFFER AMERICAN BEST SELLERS

APPLEWARE is a new range of software for the Apple II computer backed by the resources of leading British Computing Group, A.C.T.

Appleware is being launched with a catalogue of seventy-five established best sellers from seven major American suppliers of Apple software. Contributing software houses include Programma International, Personal Software, Automated Simulations, Speakeasy Software and Program Development Inc. ACT are also developing a wide range of Business packages here in this country.

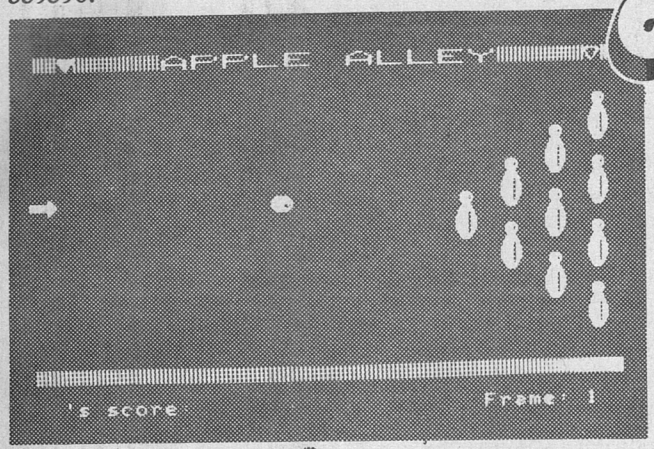
Appleware programs will be available nationwide from official Apple dealers or direct from ACT Appleware in Birmingham, by post. All programs are on disk.

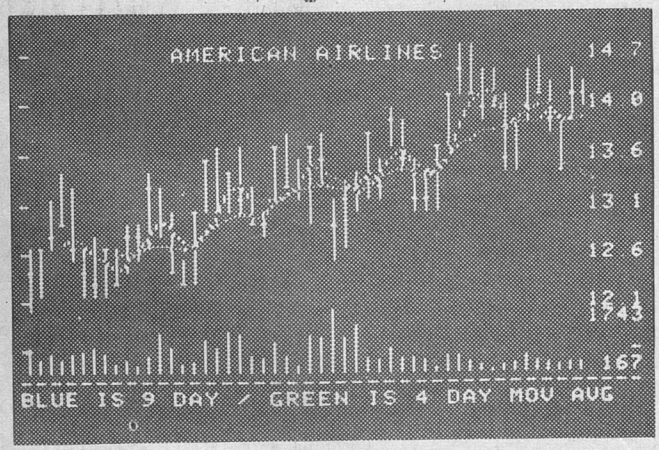
The establishement of a single distribution source for Apple software has been welcomed by dealers who believe that sales of Apple have suffered from the lack of software support.

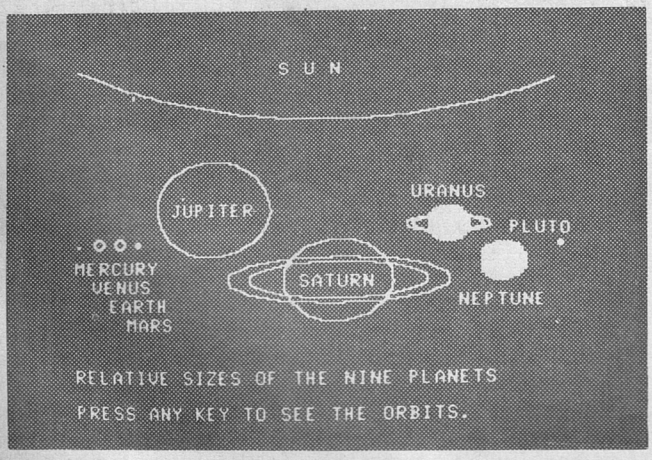
In launching Appleware ACT will be able to draw upon their experience with Petsoft, market leaders in software for the Commodore PET. Petsoft's existing network of European and overseas distributors will also be used.

Amongst titles offered in the first Appleware catalogue are Personal Software's acclaimed 'VisiCalc' pro-

gram at £125, described by Dr Adam Osbourne as "a work of art", Programma International's 'Apple Invaders' at £8 and the 'talking Disk' speech synthesis program at £14.95. The package most in demand from dealers is the 'Apple Database' retailing at £23.50.
Free copies of the Appleware catologue are available from ACT Appleware, 66-68 Hagley Road, Edgbaston, Birmingham B16 8PF, telephone 021-455 8585, telex 339396.







ACT
Appleware
Software for
Apple II Computer

Radclyffe House, 66-68 Hagley Road, Edgbaston, Birmingham
B16 8PF. Telephone: 021-455 8585 Telex: 339396

HITCH-HIKER'S GUIDE TO THE PET.

After extensive explorations within the PET, Christopher Preston of ACT has developed a Hitch-Hiker's Guide to the PET's operating system.

Originally prepared to assist Petsoft's personnel, the Guide was found so useful that it was decided to publish. The three parts of the book are Useful Locations, which describes the meaning and use of various zero page and other operating system locations, Useful Subroutines which explains the functions of several of the operating system routines including I/O, Error Detection and Reporting, and PET File Handling, which describes how to read and write data-files from an assembler program, and includes printing and other IEEE functions.

The book is intended for both BASIC and Assembler level programmers.

Christopher Preston has just been appointed Technical Manager in charge of software development at Petsoft.

The Hitch-Hiker's Guide to the PET is available at £5 from PET dealers or direct from Petsoft at 66-68 Hagley Road, Edgbaston, Birmingham B16 8PF, telephone 021-455-8585.

SCHOOLS TO RECEIVE £30,000 OF COMPUTER EQUIPMENT.

Microsense Computers, the Hertfordshire based microcomputer distributors today presented £30,000 worth of Apple Computers and accessories to selected schools throughout Britain.

In conjunction with MUSE (Mini and Micro Computer Users in Secondary Education) the seven schools listed below have been chosen to receive an Apple Computer and accessories including disc drives, video monitors and printers. A further five schools will also benefit under this scheme, but these have yet to be selected.

Tewin Water School
Digswell
Welwyn
Herts AL6 0AA
Mr Ian McEwan
The Headmaster
Tel: 043871 4288

The Netherhall School
Queen Edith's Way
Cambridge
CB1 4nn
Mr Alan Greenwell
Tel: 0223 42931

Brays School
Brays Road
Sheldon
Birmingham B26 1NS
Mr Ian Glen
The Headmaster
Tel: 021 743 5730

Plympton St Maurice Primary School
Bullers Hill
Plympton
Devon
Mr Tony Mullard
The Headmaster

Worcester College for the Blind
Whittington Road
Worcester WR5 2JU
Mr Richard C Fletcher
Tel: 0905 354627

Trent Polytechnic
Clifton
Nottingham NG11 8NS
Mr Hugh C Williams
Tel: 0602 48248 Extn 3253

Marsh Hill Comprehensive School
Hampton Road
Erdington
Birmingham B23 7JL
Mr Bob Trigger Mr Ian Drury
021 373 5868

These schools have been chosen as a cross section of educational establishments which could benefit from the use of computors in their teaching programme.

The Computers can be used both as a teaching aid and to increase awareness of the applications of computers in today's modern society.

A furtherance of these applications is one aspect of MUSE's own objectives which are to provide schools and individuals with an information and advisory service on educational software and computer equipment in general.

The formal presentation was made in London by Mr. Michael Brewer, Chairman of Microsense to Mr. John Coll, Chairman of MUSE and members of the MUSE Committee.

THE FIRST LONDON COMPUTER FAIR
July 11th & 12th 1980. 10am - 5pm.

The North London Hobby Computor Club has joined together with some of the other computer clubs in London to form the Association of London Computer Clubs. They have decided to hold a computer fair, which they are calling the London Computer Fair, on July 11/12th 1980. This will be held in the theatre of the Polytechnic of North London.

E.C.C. Publications, the publishers of Educational Computing, Business Matters and Which Computer, are sponsoring the event and Clive Sinclair has agreed to open the Exhibition.

The exhibition consists of Retail Exhibitors, Hobbyists, Club Stands, Seminars, Workshops, Surgeries, Bazaar and Jumble Sales.

SEMINAR: On Friday the 11th, Educational Computing will assist the organising of a seminar devoted to the subject 'Computers in Education'. Well known speakers in this field will be there, and, as it's the last week of term, many teachers and lecturers.

WORKSHOP: On Saturday the 12th, the Commercial User' Group of the North London Hobby Computer Club and Business Matters are arranging for 'experts' and users to be available to answer prospective commercial buyers' questions on a one-to-one basis.

BAZAAR: On Saturday also, A Bazaar of used or 'obsolete' equipment will take place in the morning. In the afternoon, there will be a Jumble Sale of software and hardware.

You can find the Polytechnic in the centre of London and is easily accessible by road, rail and bus. Accommodation is available as a number of rooms have been booked in the Hall of Residence and preferential rates are given below:

Bed and Breakfast: £5.25
including evening meal: £7.25

See you there!

BUSINESS MODEL FOR PET

The well known mainframe business simulation developed bys Understanding Limited has been converted to run on the £750 32K PET microcomputer. Known as PETPLAN, the programm creates an economic model of a medium size manufacturing company. Up to four players, acting as a board of directors, take decisions relating to production, marketing, finance and investment. Each decision has complex and far reaching consequences, the effects of which are displayed in the form of balance sheets, production and sales reports on the screen or printed out on paper.

The original mainframe program has been widely used for management training purposes by colleges and large companies. The U.K. distributors, Petsoft, however, report that they have already received a large number of enquiries from schools and small businessmen owning PETs. "with around 25,000 PETs in Britain we can afford to sell the program for only £60, a fraction of the original price", says Julian Allason of Petsoft.

PETPLAN was developed by a team lead jointly by Martin Mitchell an economist and chartered accountant and Pat Shackleton of Understanding Ltd., the London based software house who have specialised in economic modelling and business simulations for a number of years.

The options open to 'Directors' at each stage of the program are based on extensive studies of decision making in manufacturing industry. Equally realistically the results of a decision to invest in more plant or to employ more salesmen, are felt only gradually over a period of time.

The PETPLAN business model is designed to teach businessmen, trainee managers and students basic business techniques like cash flow projection and sales forecasting, and to develop management skills by the most effective method possible: experience.

The PETPLAN package includes the program on cassette, a comprehensive 50 page manual with full instructions, sample run and business forms plus a voice guide on cassette which may be played in any standard tape player. It is available from Petsoft, 66-68 Hagley Road, Edgbaston, Birmingham B16 8 PF (telephone 021-455-8585 or from PET computer dealers throughout the country.

LIFE FOR THE APPLE II/ITT 2020

A machine code version of one of the world's most famous computer games - 'LIFE 1' from Computercraft Ltd. Written to run on the APPLE II or ITT2020 microcomputers, 'LIFE 1' offers the user a variety of attractive features.

DOCUMENTATION: The program comes with a comprehensive guide to its use, and the program itself contains a HELP facility for fast reference while running.

SPEED: The use of machine code throughout 'LIFE 1' ensures maximum speed of execution and response - up to three generations or frames of Life can be displayed per second. It also ensures that the program will run on a machine with only 16K of main memory.

EASY PATTERN ENTRY: Entering and clearing Life patterns on the screen is fast and efficient, and at the end of a simulation the user has the option of recalling the original pattern for modification.

FULLY CONTROLLABLE DISPLAY: Having commenced Life generation, the user may slow the simulation or run it in single steps. The display can be stopped at any point, altered and restarted.

PATTERN STORAGE AND RECALL: While running the program, the user may store and recall up to three screenfulls of Life simulation in reserved areas of memory. The program also offers the capability to save interesting simulations on cassette.

SHAPE LIBRARY: The program contains a library of many of the most well known Life shapes (e.g. glider gun, spaceships, barber pole, etc.) These may be displayed on the screen in any desired position or orientation.

'LIFE 1' also offers the user a choice of wrap-around or bounded universes, and a continuous display showing the number of elapsed generations and the on-screen population.

The program comes on high-quality cassette tape (verified) with an accompanying explanatory booklet. The running requirements are for an APPLE II or ITT 2020 with 16 Kbytes or more of RAM. 'LIFE 1' is available by mail order at £9.00 per copy including VAT and post and packing from:

Computercraft Ltd.,
129, Newington Green Road,
London N1 4HA.
Phone : 01-226-0656.

MICROELECTRONICS DEVELOPMENT PROGRAMME FOR SCHOOLS AND COLLEGES

The Government today announced a £9 millions microelectronics development programme for schools and colleges. In answer to a Parliamentary Question in House of Commons from Mr Tam Dalyell, Mp for West Lothian, mr Neil Macfarlane, Parliamentary Under Secretary of State at the Department of Education and Science, said:

"With the agreement of his colleagues my Rt Hon and learned Friend has decided to initiate a microelectrics development programme for England, Wales and Northern Ireland starting in 1980/1. A separate programme is being mounted in Scotland. The programme wiil give schools and colleges a better understanding of the potential applications of microelectronic technology by commissioning new development projects and by building on existing work in this field. £9 millions at November 1979 prices will be made available for this purpose over the next four years."

£1 million will be made available in 1980/81 and approximately £2.7 millions in each of the three succeeding years.

The programme will serve the needs of education and industry by helping schools and colleges to make better use of microelectronics as a teaching resource and to equip young people with the skills which are required to exploit the economic potential of the "new technology". Much innovation is already taking place and the programme will only be successful if local education authorities take up the ideas and activities which prove to be effective. It is designed to stimulate and support local initiatives and not to replace them.

The programme will consist of a number of co-ordinated projects. The projects will be commissioned through national bodies, local organisations and individual institutions. The programme will be managed by a full time director under the supervision of the educ -ation departments and in the light of expert advice. Projects will be financed with grants from the education departments. The Council for Educational Technology will be invited to provide central, administrative and accounting services.

The education departments have already received a large number of proposals for work in this field and the Secretaries of State are inviting interested organisations to submit fresh ideas so that these too can be considered for support. It is hoped that a number of projects can be started in 1980/81.

The aim will be to secure a balanced programme which will complement existing activities including those financed independently and through other Government agencies. The following general guidelines are being issued to assist those preparing proposals under the programme:

★ Priority will be given to curriculum development, to the development of material for teacher training courses, and to arrangements to make more and better software available to schools and colleges.

★ Particular attention will be paid to curriculum development in subjects such as science, mathematics, craft design and technology, business studies and technician studies. There is also scope for work in the field of special education.

★ A successful outcome will depend on the extent to which promising ideas are developed beyond the life of the programme. Projects should aim to stimulate work which can be undertaken within the resources of local education authorities. Wherever possible, local education authorities and institutions should be invited to commit their resources to projects alongside those provided from programme funds.

Arrangements are being made to appoint a director as soon as possible. In the interim, proposals should be sent to Schools III Branch, Department of Education and Science, Elizabeth House, York Road, London SE1 7PH. Proposals for 1980/81 should be submitted as soon as possible.

ACORN ATOM

THE case has been designed to be stylish but harmonious in self coloured heaving duty 70% density polystyrene. The keyboard used is a full sized QWERTY layout with special function keys for screen editing, delete, control etc. The key swiches are electronically de-bounced and have gold plated contacts tested to 3 million operations. The key buttons are arranged in stepped lines and have standard travel and light pressure action equally suitable for fast touch typing and hesitant one finger operation.

The entire unit weighs approximately 3.5lb and its soft rubber feet give a reassuringly firm grip on most surfaces.

All that is required for immediate operation is the ATOM mains power supply or suitable equivalent and any U.H.F. television. Input to the television is via the aerial socket and with the set tuned to channel 36, a jitter free picture with clear legible characters is produced.

Internally components are mounted on a top quality fibre printed circuit board with sockets for all high cost integrated circuits and optional devices. The ATOM has been designed for enormous expansion on the circuit board but for the purposes of producing a thoroughly practical computer at very low cost the basic unit is provided with 2 kilobytes of random access memory, allowing ½K for display, 1K for the system and ½K for programs, although display memory may also be used for programs. The basic system has 8K of read only (fixed) memory which holds the BASIC interpreter and ATOM Operating System (O.S.) ATOM's BASIC has all normal integer BASIC functions plus many powerful extensions (such as DO-UNTIL and labels) usually associated with higher level languages. ATOM's 32 bit arithmetic allows much higher accuracy than most BASICS and numbers up to + 2,000,000,000 may be used compared to + 32,000 with 16 bit arithmetic. In the minimal system graphics resolution is 64 x 48 in black, white, and grey and a wide range of plot commands is available in BASIC. Another important extension is a mnemonic assembler which allows the user to write machine code programs, and assembler may be used within BASIC programs to combine the speed advantages of machine code with the convenience of a high level language. Audible error indication is given by an internal loudspeaker which can also be used under program control for the creation of musical sounds.

Simply inserting chips into sockets on the printed circuit board allows the large scale expansion which is fundamental to the design concept. Thus the total RAM may be increased to iiK allowing high resolution (256 x 192) graphics, two further ROM chips can be added, for example the Acorn Floating Point scientific ROM plus a utility package, totalling 16K on board. In addition to 5 I/0 lines partly used by the cassette interface an optional VIA device provides a very wide variety of use I/0 and timer functions, and via a buffer device allows direct printer drive controlled by the standard ROM. An extra module used output from the standard board to produce red green and blue signals for a colour monitor or colour t.v. An in board connector strip takes the ATOM communications loop interface, with this module inside; the case any number of ATOMs may be linked via three wires to each other or to a master system with, for example, mass storage/hard copy facility. The Acorn bus is taken out at the rear of the case and may be connected to any number of the Acorn range of cards, and in addition any **one** Acorn card may be fitted internally.

Hardware	**Technical Description**
Memory:	From 2K to 11K RAM on board (in steps to 1K) up to 40K including external memory.
	From 8K to 16K ROM (two 4K additions)
Processor:	6052 with 1 Mc/s clock
Video Display Generator :	6847 generates video signals for 8 different modes including: high resolution graphics (256 x 192), Red green, and blue graphics up to a resolution of 128 x 192, and mixed ASCII characters and semi graphics.
PIA :	8255 provides keyboard scan, cassette I/0, audio output, graphics control and user I/o.

VIA :	6522 provides two 8 bit + control I/0 ports (one used for printer output) plus a wide range of serial I/0 functions and dual timers.
Cassette INTERFACE	CUTS 300 baud, involves minimum hardware (zero crossing detector input and output from timer) to allow user to redefine tape routine to virtually any speed or standard.
Loudspeaker :	Driven from 8255 via buffer allowing software tone generation of any frequency.
B/W Video Output :	To monitor
UHF Modulator Output :	Channle 36 domestic T.V.
Bus Output :	Fully buffered address and data bus plus internal connections for one Acorn Eurocard.
Power Requirement :	Minimum system: 8 volts @ 800 mA (from ATOM power unit feeding internal regulator). Maximum system: 5V @ 1.8A from external regulator supply.

Software

ATOM BASIC	32-bit arithmetic (+ 2,000,000,000) High speed execution 43 standard and extended BASIC commands Variable length strings (up to 256 characters) String manipulation functions 27 32-bit integer variables 27 additional arrays Random number function PUT and GET bytes, words and strings to and from files WAIT command for timing DO-UNTIL construction Commands may be abbreviated for economy Multiple statements per line Logical operators (AND, OR, EX-OR) LINK to machine code routines Numbers can be input and printed in hexadecimal Symbolic labels for fast branches and subroutine calls Powerful indirection operators (?,!) Graphics facilities to draw points and lines 16 PLOT commands, MOVE and DRAW
ASSEMBLER :	Mnemonic Assembler for machine code programming Formatted listing Assembler and BASIC may be combined Standard 6502 mnemonics Provides symbols, automatic resolution of forward references Macro-facilities Breakpoints may be inserted for debugging
VDU :	32 characters x 16 lines Inverted characters Automatic scrolling Paged / Non paged modes All control codes can be generated Screen editing

Operating System :	CUTS cassette routines with checksum Filenames up to 12 character LOAD and SAVE BASIC and assembler programs or text files Search (catalogue) routine Software hook to optional disc drive and communication loop modules Printer drive routines
Optional Maths Software :	Floating point maths functions to 9 digit accuracy including arithmetic, trigonometric and hyperbolic functions.
Optional Communication Software :	Allow high speed bi-directional interface to other ATOMs or peripherals. Allows transfer of control or data to other modules in loop.
Optional Utility ROM :	Such as the ONLIBASIC extension for real time control of laboratory experiments.

Each ATOM is provided with a comprehensive manual written on two levels, with a teach yourself BASIC section for those totally unfamiliar with computer programming, and a language reference section giving a complete summary of the ATOM's facilities. Another section on the Assembler and Cassette operating system gives a guide to machine code programming with the 6502. All sections are fully illustrated with example programs including: three dimensional graph plotting, animated displays, audio visual games, accounting, arbitrary precision arithmetic etc.

When the ATOM is purchased in kit form, full assemblv intstructions are provided which include hardware tests for fault finding. All components are guaranteed and full service/repair facilities are available at our Cambridge laboratories.

Price (excluding V.A.T.)

Standard ATOM Kit - £120

Standard	ATOM	assembled	£150
Standard	Power	Unit	£10

Post and packing (UK) - £2

THE NEED FOR ALL CHILDREN TO HAVE A BASIC KNOWLEDGE OF COMPUTERS

THE new technology has implications for Britain's future which are quite immeasurable, but, which will affect the future planning of every single Government Department. The widespread use of computers will alter the life style of all our people to a great or lesser extent, though the most difficult thing to forecast is the time scale involved.

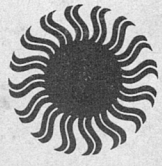
Much coverage has been devoted to the employment factor and whether we agree with Professor Stonier that most jobs will disappear for good as a result of the new technology, or whether we prefer to believe, with Rothschild, that opportunities for new jobs will emerge, there is no doubt that today's school leaver must expect changes during his or her working lifetime. It is probably true that the job he or she first obtains will cease to exist at some indeterminate time in the future and he or she will have to look for another one, perhaps one directly produced by the new technology, or perhaps brought into existance because routine tasks have been taken over by machines, thus releasing manpower for jobs involving personal services or needing a greater human input.

I do not believe any Government Department has yet started to come to terms with future change. Indeed, political decision-makers are even less well prepared for these changes as a result of the abolition of the House of Commons Select Committee on Science and Technology.

I welcome the announcement from the Department of Education and Science that funds are to be made available to promote education in micro-electronics. Not every child is going to be a computer programmer, but, I believe that no child should leave school without a basic understanding of what computers can do and equally important, what they cannot.

I see this as merely a beginning and despite the shortage of public funds, which currently face us, it is vital that the Government should be pressed to devote greater resources to this vital area.

Talk given by: Mr G WALLER,
MEMBER OF PARLIAMENT FOR BRIGHOUSE AND SPENBOROUGH to The Computer Retailers Association on 12th March, 1980



How to get your **LIVERPOOL SOFTWARE GAZETTE** regularly

Imagine the disastrous effect on your life style if you missed a single issue. The possibility of trauma is easily eliminated by the simple expedient of acquiring a regular subscription at the all time bargain price of £6.00 for the next twelve scintillating issues.

Don't miss the chance of a lifetime
Fill in the form below.

Commercial and educational organisations requiring a large number of copies can buy in bulk at advantageous rates from:

Computer Bookshop,
Temple House,
43-48 New Street
Birmingham B2 4LH

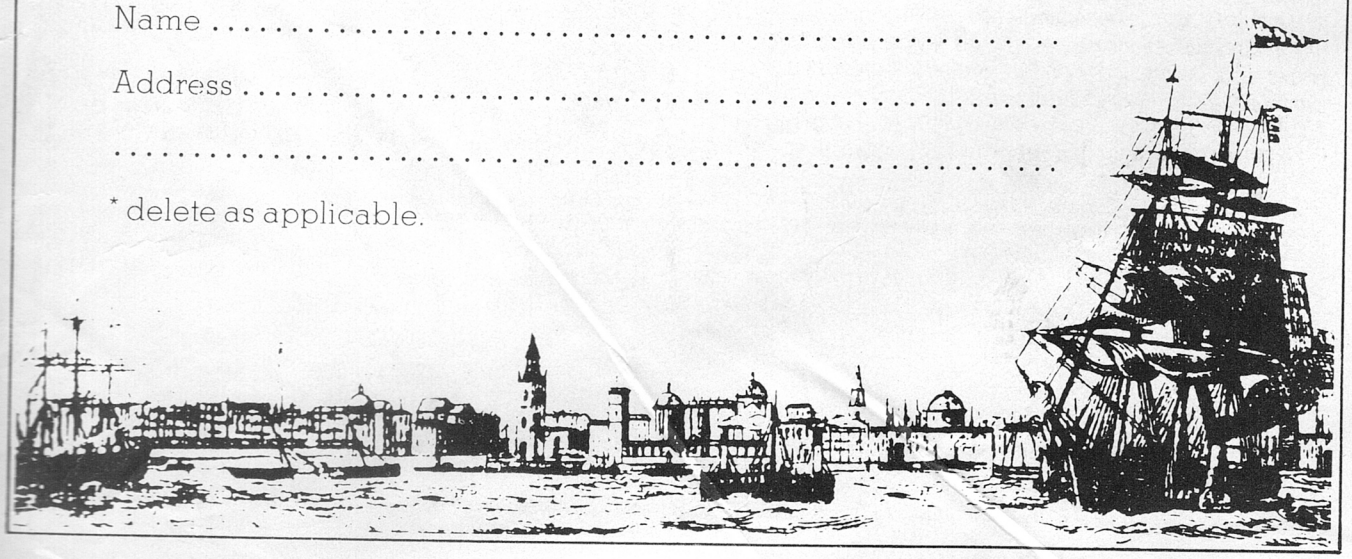
Please send me twelve issues of the Liverpool Software Gazette starting with the first/second/third/fourth/fifth issue. Cheques and PO's should be made payable to Liverpool Software Gazette and sent to us at 14 Castle Street, Liverpool.

Name .....

Address .....

.....

* delete as applicable.



Unique in concept - the home computer that grows as you do!

New! - The Acorn Atom

£120 plus VAT and p&p.

An outstanding personal computer kit

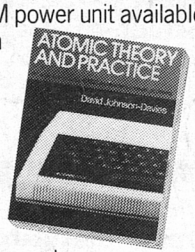
Also available ready-built **£150** plus VAT and p&p

The ATOM – a definitive personal computer. Simple-to-build, simple-to-operate. But a really powerful full-facility computer. And designed on an expandable basis. You can buy a superb expanded package now – tailored to your needs. Or, you can buy just the standard Atom kit, and, as you grow in confidence and knowledge, add more chips. No need to replace your equipment. No need to worry that your investment will be overtaken by new technology. As you need more power, more facilities, you can add them!

*The picture shown demonstrates mixed graphics and characters in three shades of grey provided by the Standard Atom.

The standard ATOM kit includes:

● Full sized QWERTY keyboard ● Rugged polystyrene case ● Fibreglass PCB ● 2K RAM ● 8K ROM ● 23 integrated circuits ● Full assembly instructions including tests for fault-finding. (Once built, connect it to any domestic TV and power source) ● Power requirement: 8V at 800 M A. ATOM power unit available. See coupon. PLUS FREE MANUAL written in two sections – teach yourself BASIC and machine code for those with no knowledge of computers, and a reference section giving a complete description of the ATOM's facilities. All sections are fully illustrated with example programs.



The ATOM concept

Adding chips into sockets on the PCB allows you to progress in affordable steps to large-scale expansion. You can see from the specifications that the RAM can be increased to 12K allowing high resolution (256 x 192) graphics. Two further ROM chips, e.g. maths functions, can be added directly to the board giving a 16K capacity. In addition to 5 I/O lines partly used by the cassette interface, an optional VIA device can provide varied I/O and timer functions and via a buffer device allow direct printer drive. An optional module provides red, green and blue signals for colour. An in-board connector strip takes the ATOM communications loop interface. Any number of ATOMs may be linked to each other – or to a master system with mass storage/ hard copy facility. Interface with other ACORN cards is simplicity itself. Any one ACORN card may be fitted internally.

So you can see there are a vast number of modular options and additions available, expanding with your ability and your budget.

The ATOM hardware includes:

● Memory from 2K to 12K RAM on board (up to 35K in case) ● 8K to 16K ROM (two 4K additions) ● 6502 processor ● Video Display allows high resolution (256 x 192) graphics and red, green and blue output ● Cassette Interface – CUTS 300 baud ● Loudspeaker allows tone generation of any frequency ● Channel 36 UHF Modulator Output ● Bus output includes internal connections for Acorn Eurocard.

The ATOM software includes:

● 32-bit arithmetic (±2,000,000,000) ● High speed execution ● 43 standard/extended BASIC commands ● Variable length strings (up to 256 characters) ● String manipulation functions ● 27 32-bit integer variables ● 27 additional arrays ● random number function ● PUT and GET byte ● WAIT command for timing ● DO-UNTIL construction ● Logical operators (AND, OR, EX-OR) ● LINK to machine-code routines ● Plot draw and move.

ACORN COMPUTER

4a Market Hill, CAMBRIDGE CB2 3NJ

Your ACORN ATOM may qualify as a business expense. To order complete the coupon below and post to Acorn Computer for delivery within 28 days. Return as received within 14 days for full money refund if not completely satisfied. **All components are guaranteed with full service/repair facility available.**

Please send me the following items:

Quantity	Item		Item price inc. VAT+p&p	TOTALS
	ATOM KIT 8K + 2K (MIN)		@ £140.00	
	ATOM ASS 8K + 2K (MIN)		@ £174.50	
	ATOM KIT 12 + 12K (MAX)		@ £225.00	
	ATOM ASS 12 + 12K (MAX)		@ £289.50	
	1K RAM SETS		@ £11.22	
	4K FLOATING POINT ROM		@ £34.80	
	PRINTER DRIVE	6522 VIA	@ £10.65	
		LS 244 Buffers (pair)	@ £3.17	
	MAINS POWER SUPPLY (1.5 amps)		@ £10.20	
			TOTAL	

To: Acorn Computer Ltd., 4a Market Hill, CAMBRIDGE CB2 3NJ

I enclose cheque/postal order for £_____

Please debit my Access/Barclaycard No. _____

Signature _____

Name (Please print) _____

Address _____

Telephone No. _____

Access

BARCLAYCARD VISA

Registered No: 1403810. VAT No: 215 400 220

LSG/5/80